Iterative Methods for Earthquake Cycle Simulations with HPC Applications

Alexandre Chen, University of Oregon

***Abstract*—High Performance Computing (HPC) is a powerful tool in scientific research for experimental simulation and data analysis. However, it has not been well studied and applied for earthquake cycle simulation. In this work, we explore computational methods and HPC techniques for earthquake cycle simulation with a focus on iterative methods and preconditioning techniques. The problem that we target is solving Poisson's equation numerically with boundary conditions derived from geophysics. Specifically, we use summation-by-parts (SBP) finite difference operators with simultaneous-approximation-term (SAT) method. The combined SBP-SAT scheme has high-order accuracy and provable stability for our problem. The purpose of this work is to examine existing research on problems similar to ours, so we can design algorithms for a large-scale 3D earthquake cycle simulation suitable for HPC platforms in the future. We begin with a mathematical introduction to our problem, and then iterative methods for solving this problem numerically. Finally, we explore the HPC aspects of these different iterative methods.**

I. Introduction

In mathematics, a partial differential equation (PDE) is an equation that describes the relationship between various partial derivatives of a multivariable function. PDEs are widely used to describe laws in science and engineering. Analytical solutions to these PDEs are often hard or impossible to obtain. In these cases, numerical methods, such as finite difference methods (FDM) or finite element methods (FEM), are often applied to obtain numerical solutions to these PDEs. Numerical discretizations to PDEs give rise to a multitude of linear algebra operations including matrix-vector products and linear and non-linear solves (i.e. solving systems of equations). SBP-SAT methods are used for approximating solutions to PDE, but so far the application domains remain largely confined to serial code and small problems. Their ubiquity in modern scientific computing applications, however, means their development for HPC would have much larger impacts in many scientific fields.

We are particularly motivated by the simulations of the earthquake cycles, where long quiescent periods are characterized by static deformation and a wide range of spatial scales (due to mm-scale frictional effects and km-long faults) give rise to large-scale problems [28]. High-performance code for these long-term earthquake dynamics is still in its infancy, due to the somewhat daunting reality that the static solution (i.e. solving $Ax = b$) involves a huge computational cost. Simulations of earthquakes in 3D volumes involve upwards of $\sim 10^9$ degrees of freedom. Even though the matrix A is sparse (i.e. most entries of the matrix are zeroes), factoring the matrix has a significant cost in terms of floating point operations and memory. In addition, direct methods for solving a large linear system (e.g., matrix factorization) in parallel are challenging for many reasons.

Iterative methods, on the other hand, use an initial value to generate a sequence of improving approximate solutions to the original problem. There are different types of iterative methods suitable for different types of problems. Traditional iterative methods like the Jacobi method and the Gauss-Seidel method are suitable for linear systems whose matrices have a spectral radius of less than one. Modern Krylov subspace methods such as Conjugate Gradient (CG) are suitable for linear systems with positive definite matrices. [17] These different iterative methods have various properties in computation. Gauss-Seidel is known to have reduced memory requirements compared to the Jacobi method, and CG has much faster convergence for positive definite matrices compared to the Gauss-Seidel method. In practice, we need to consider different properties of our linear system as well as the computational platforms and environments when choosing the optimal iterative solver for our problems. Another nice advantage of using iterative methods compared to direct methods is that they do not require explicit matrix formulation. Instead, a matrix-free function can replace matrix-matrix or matrix-vector multiplication used in these methods, which can reduce the memory requirement for these methods and reduce data input and output (I/O) significantly. The latter property is vital when using architectures with high latency in data I/O such as graphics processing unit (GPU) for the HPC purpose.

II. HPC in Earthquake Simulations

A. Earthquake Cycle Simulation

Earthquake cycle simulations with laboratory-derived friction laws have been studied to simulate sequences of large-scale historical earthquakes. Recent work includes a large-scale simulation of earthquake occurrences along the Nankai Trough in southwest Japan [67], and the simulation of the recurrence of two asperity ruptures including the afterslip which triggers the other earthquake in the Sanriku region in northeast Japan [49]. In these papers, the authors studied the large-scale heterogeneity of parameters or asperities in friction laws. Another approach is to consider microscopic structures, examining the earthquake occurrences on a large fault with different levels of heterogeneity to produce seismic patterns with Gutenberg-Richter statistics resembling those natural earthquakes [43]. Meanwhile, long-term and short-term slow slip events, as well as low-frequency events have been observed on the plate interfaces in the deep extended regions of seismogenic zones [44]. Furthermore, recent simulations have

demonstrated the possibility that the activity of such events could change before the occurrence of large interplate earthquakes [5]. To achieve realistic earthquake cycle simulations, it is necessary to develop large-scale models of earthquake cycle simulation that capture the earthquake mechanisms for these different regions with multiple scales. However, it is difficult to achieve this kind of multi-scale simulation in larger regions without HPC techniques.

B. HPC in Geophysics Simulation

HPC is not something novel in geophysics. Traditionally it has been used in seismic data processing for understanding earthquake mechanisms [31] or imaging underground geological structures to search for hydrocarbon deposits [80]. However, these applications mainly involve data analysis instead of computational simulation. More recent progress was made in simulating earthquake impacts and hazard prevention. The integrated earthquake simulation (IES) is used to seamlessly simulate three earthquake processes, namely, the earthquake hazard process, the earthquake disaster process, and the anti-disaster action process. [78]. HPC is essential to IES if the simulation is done in an urban area where $10^4 \sim 10^6$ structures are located. This paper applied HPC to the earthquake disaster and hazard estimation for the urban area in Tokyo metropolis [45]. These simulations and related work were done to study the post-earthquake response from architectural structures to urban management using various methods from geophysics to computational sociology. Similar work includes the study of the tsunami generated from an earthquake using HPC based on laboratory-derived physics laws [84]. However, the applications of HPC directly to the study of earthquake nucleation which can help explain how earthquake starts are minimal. Results from such research would help with hazard prevention better in addition to post-earthquake response in traditional studies. The most related research in this sense was using hierarchical matrices (H-matrices) for fast computation of quasi-dynamic earthquake cycle simulation [67]. Due to the limitation of the computational capability available, the authors used hierarchical matrices to calculate approximations for multiplicative computations of $N \times N$ slip response function matrix and slip deficit rate vector, where N is the number of divided cells on the plate surface. The N used in this paper ranges from 10^4 to 10^6, representing roughly 100 to 1000 cells along one direction in a 2D surface. By using H-matrices, the computational time and memory size in earthquake cycle simulations were greatly reduced, so large-scale simulations were possible. The drawback is also obvious, using H-matrices only provides a lower-rank approximation to the original matrices without proven convergence or numerical stability. This also differs from finite difference methods where the discretization of a real domain is adopted. Therefore, this can not be used for the SBP-SAT method. Thus, we need to look for other HPC approaches to apply the SBP-SAT method to our research. Also, since this work was done, the computational capabilities of modern supercomputers have increased exponentially. This allows us to work on a much finer resolution that can simulate earthquake cycles better. However, the increased amount of data and computation compared to this work has raised several HPC challenges in problem formulation and algorithm implementation that weren't encountered with reduced computation and memory from approximation.

III. Elliptic Partial Differential Equation and Finite Difference Method

A. Elliptic PDEs

1) Governing Equations: Second-order linear partial differential equations are classified as either elliptic, hyperbolic or parabolic depending on the coefficients of the PDEs. In general, second-order linear PDEs with two variables can be written in the following form

$$Au_{xx} + 2Bu_{xy} + Cu_{yy} + Du_z + Eu_y + Fu + G = 0 \quad \text{(III.1)}$$

where A, B, C, D, E, F, G are functions of x and y, and $u_x = \frac{\partial u}{\partial x}, u_{x,y} = \frac{\partial^2 u}{\partial x \partial y}$ and similarly for u_{xx}, u_y, u_{yy}. A PDE written in this form is called elliptic if

$$B^2 - AC < 0 \quad \text{(III.2)}$$

The simplest example of elliptic PDEs is Poisson's equation $\Delta u = u_{xx} + u_{yy} = f(x, y)$, where $f(x)$ is called the source term. When $f(x) = 0$, there is no source term in this equation. This simplified version of Poisson's equation is called Laplace's equation. Poisson's equation has been used in a lot of different fields. For example, the solution to Poisson's equation is the potential field caused by a given electric charge or mass density distribution.

Poisson's equation also describes the static deformation of the solid Earth over the time scales of earthquake cycles for the equilibrium state. The standard assumption is that the Earth is linear elastic, giving rise to the PDE

$$(\lambda + \mu)\nabla(\nabla \cdot \mathbf{u}) + \mu\nabla^2\mathbf{u} + \mathbf{f} = \mathbf{0} \quad \text{(III.3)}$$

defined on a sub-domain of $\mathbb{R}^3$. Here λ and μ are material parameters (Lamé's first parameter and the shear modulus, respectively), $\mathbf{u}$ is the vector of particle displacements, and $\mathbf{f}$ encompasses source terms.

Due to the computational complexity of Poisson's equation in 3D with fine spatial discretization for stability, the problem is often reduced to 2D to satisfy the current computational tools available. We assume the motion of the Earth is antiplane shear, where only one non-zero component of the displacement vector exists, depending only on two spatial variables, giving rise to the 2D Poisson equation [58].

2) Initial Boundary Value Problem: In mathematics, initial values and boundary conditions are a set of additional constraints associated with a PDE to ensure that the solutions to a PDE also satisfy the settings of a problem. A PDE formulated with these constraints is called an initial boundary value problem (IBVP) [7]. An IBVP is called well-posed if it has a unique solution. The discretization of a such problem results in an invertible matrix that gives a unique solution to the linear system. Fourier analysis is a powerful tool for solving these IBVPs with analytical solutions if the problem is well-posed with simple initial and boundary conditions[7]. For real

problems with more complex initial and boundary conditions, numerical methods are used to obtain numerical solutions. There are mainly two types of numerical methods for solving PDEs, finite difference methods (FDM) and finite element methods (FEM). We will be focusing on the finite difference methods in subsection III-B as our numerical method for discretization is a type of FDM. Still, we will also cover some basic concepts of FEMs that are widely used for geophysics study, especially handling meshes with complex geostructures [14, 24, 50].

3) Types of Boundary Conditions: There are many different types of boundary conditions in IBVP. We summarize them in the following table Table I.

Name	1st part of boundary	2nd part of boundary
Dirichlet	$u = f$	
Neumann	$\frac{\partial u}{\partial n} = f$	
Robin	$c_0 u + c_1 \frac{\partial u}{\partial n} = f$	
Mixed	$u = f$	$c_0 u + c_1 \frac{\partial u}{\partial n} = f$
Cauchy	$u = f$ and $\frac{\partial u}{\partial n} = g$	

TABLE I: Summary of boundary conditions for the unknown function, u, constants c_0, c_1 specified by the boundary conditions, and known scalar functions f and g specified by the boundary conditions.

B. Finite Difference Methods

In numerical analysis, finite-difference methods are a class of techniques for solving differential equations by approximating derivatives with finite differences. The discretization is applied to both spatial domain and time interval (if the problem is time-dependent) where they are broken into a finite number of steps, and the value of the solution at different discrete points is approximated via solving algebraic equations with finite differences and values from neighboring points. Ordinary differential equations or partial differential equations are converted into linear or nonlinear systems via finite difference methods. These systems can be solved using methods in linear algebra. Particularly, modern computers are very efficient for calculations in these methods.

The finite difference approximations are derived from Taylor series expansions. The orders and accuracy of different finite differences are related to the order of the remainder term in the Taylor series expansions, and they play an important role in the accuracy of the finite difference methods as well as in the convergence rate when using iterative methods to solve the linear or nonlinear system. In this work, we focus on solving linear systems that finite difference methods give rise to.

When solving IBVPs with finite difference methods, how to enforce boundary or initial conditions in Table I is the main challenge. There are two ways to enforce boundary conditions. Strong enforcement of boundary conditions or injection is used in traditional finite difference methods [16]. Because symmetric positive definiteness is an important condition for many iterative solvers, additional techniques might be used in strong enforcement to form a symmetric positive definite matrix. For example, when enforcing Neumann boundary conditions, it is common to divide the rows in the linear system for the Neumann boundary conditions by half [16].

Another way to enforce boundary conditions is to use weak enforcement. This approach is important for the summation-by-parts finite difference methods introduced in the next section.

C. Finite Element Method

The finite element method (FEM) is another way to solve partial differential equations in two or three space variables by subdividing a large system into smaller, simpler parts that are called finite elements. Finite element methods use weak formulations to construct sub-problems on each finite element. A finite element model of a problem gives a piecewise approximation to the governing equations. The solution of a finite element method is a combination of continuous solutions vs the discrete solutions only for grid points in an FDM.

FEMs usually have an accurate representation of complex geometry and the inclusion of dissimilar material properties. The total solution usually has an easier form than FDM. However, FEMs usually require a large amount of data for nodal connectivity information in the mesh, and higher computational costs than FDMs for complex problems[47, 42].

IV. SBP-SAT Method

A. SBP-SAT Method

Summation-By-Parts (SBP) finite difference methods have been proposed to solve problems with complex geometries such as the problem in this paper due to their desirable properties of high order accuracy and provable stability [52, 53, 82, 62]. The inter-block coupling conditions can be enforced weakly using the Simultaneous-Approximation Term (SAT) method [19, 20]. The SAT term here is analogous to the penalty term in discrete Galerkin methods.

B. One Dimensional SBP Operators

We discretize the domain $0 \leq x \leq 1$ with $N+1$ evenly spaced grid points $x_i = ih, i = 0, \ldots, N$ with spacing h = 1/N. We then project a function u onto the computational grid to be $\boldsymbol{u} = [u_0, u_1, \ldots, u_N]^T$. $\boldsymbol{u}$ is often taken to be the interpolant of u at grid points. We define grid basis vector e_j to be a vector with value 1 at grid point j and 0 for the rest. We only need e_0 and e_N to form projections at boundaries. Note that in general we have $u_j = e_j^T \boldsymbol{u}$.

We apply the class of high-order accurate SBP finite difference methods for first-order derivatives which were introduced in [52] and [53] and [82] as mentioned above. For second-order derivatives, we apply [62], with variable coefficients treated in [60]. The exact forms of definitions are given below.

Definition 1 (First Derivative)**.** *We define matrix $\boldsymbol{D}_x$ to be an SBP approximation to $\partial u/\partial x$ if it can be decomposed as $\boldsymbol{H}\boldsymbol{D}_x = \boldsymbol{Q}$ with $\boldsymbol{H}$ being symmetric positive definite and $\boldsymbol{Q}$ satisfying $\boldsymbol{u}^T(\boldsymbol{Q} + \boldsymbol{Q}^T)\boldsymbol{v} = u_N v_N - u_0 v_0$.*

Here, we only consider diagonal-norm SBP, i.e. finite difference operators where H is a diagonal matrix and D_x is the

standard central finite difference matrix in the interior which transitions to one-sided at boundaries. The condition of Q defined above can be written as $\boldsymbol{Q}+\boldsymbol{Q}^T=\boldsymbol{e}_N\boldsymbol{e}_N^T-\boldsymbol{e}_0\boldsymbol{e}_0^T$.

The reason why the operator D_x is called SBP is that it mimics the integration-by-part property

$$\int_0^1 u\frac{\partial v}{\partial x}+\int_0^1\frac{\partial u}{\partial x}v=uv\Big|_0^1, \tag{IV.1}$$

in a discrete form

$$\boldsymbol{u}^T\boldsymbol{H}\boldsymbol{D}_x\boldsymbol{v}+\boldsymbol{u}^T\boldsymbol{D}_x^T\boldsymbol{H}\boldsymbol{v}=\boldsymbol{u}^T\left(\boldsymbol{Q}+\boldsymbol{Q}^T\right)\boldsymbol{v}=u_Nv_N-u_0v_0. \tag{IV.2}$$

Following the same pattern of the first derivative, we can define the second derivative.

Definition 2 (Second Derivative)**.** *We define matrix $\boldsymbol{D}_{xx}^{(c)}$ to be an SBP approximation to $\frac{\partial}{\partial x}\left(c\frac{\partial u}{\partial x}\right)$ if it can be decomposed as $\boldsymbol{H}\boldsymbol{D}_{xx}^{(c)}=-\boldsymbol{A}^{(c)}+c_N\boldsymbol{e}_N\boldsymbol{d}_N^T-c_0\boldsymbol{e}_0\boldsymbol{d}_0^T$ where $\boldsymbol{A}^{(c)}$ is symmetric positive definite and $\boldsymbol{d}_0^T\boldsymbol{u}$ and $\boldsymbol{d}_N^T\boldsymbol{u}$ are approximations of the first derivative of u at the boundaries.*

Similarly, the operator $\boldsymbol{D}_{xx}^{(c)}$ mimics the integration-by-parts property

$$\int_0^1 u\frac{\partial}{\partial x}\left(c\frac{\partial v}{\partial x}\right)+\int_0^1\frac{\partial u}{\partial x}c\frac{\partial v}{\partial x}=uc\frac{\partial v}{\partial x}\Big|_0^1, \tag{IV.3}$$

in a discrete form

$$\boldsymbol{u}^T\boldsymbol{H}\boldsymbol{D}_{xx}^{(c)}\boldsymbol{v}+\boldsymbol{u}^T\boldsymbol{A}^{(c)}\boldsymbol{v}=c_Nu_N\boldsymbol{d}_N^T\boldsymbol{v}-c_0u_0\boldsymbol{d}_0^T\boldsymbol{v}. \tag{IV.4}$$

As noted above, we only consider diagonal-norm SBP finite difference operators here. In the interior, the operators use the minimal bandwidth central difference stencil and transition to one-sided boundaries in a manner that preserves the SBP property.

It has been known that for SBP operators defined above, if the interior operator has an accuracy of $2p$, then the interior stencil bandwidth is $2p+1$ and the boundary operator has an accuracy of p. If we use operators with interior accuracy $2p=2,4$, and 6, the expected global order of accuracy is the minimal of $2p$ and $p+2$ as evidenced by empirical study ([61] [89]) and proved for the Schrödinger equation [66].

C. 2D SBP Operators

2D SBP operators can be developed by applying the Kronecker product. For example, we discretize the 2D square domain $(x,y)\in[0,1]\times[0,1]$ 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 $(x_i,y_j)=(ih,jh)$ for $0\le i,j\le N$ with $h=1/N$. 2D SBP operators for the second partial derivatives are thus obtained from the 1D SBP operators, namely

$$\frac{\partial^2}{\partial x^2}\approx\boldsymbol{I}\otimes\boldsymbol{D}_{xx},\quad\frac{\partial^2}{\partial y^2}\approx\boldsymbol{D}_{yy}\otimes\boldsymbol{I}, \tag{IV.5}$$

see [51] for more details. We should note that Kronecker products are used here mainly for the purpose of simplicity in theoretical analysis. In computer memory, data are stored in a one-dimensional array. Hence, the Kronecker products here mainly affect the order in which we read data from a 1D array.

D. SAT Penalty Terms

SBP operators compute approximations to derivatives at all grid points, as opposed to traditional finite difference methods that only do this at interior nodes, and inject boundary data (i.e. overwrite the grid function at the boundaries to strongly enforce boundary conditions). SAT terms weakly enforce boundary conditions, penalizing the grid point at the boundary towards the boundary data. In 1D for example, if a Neumann boundary condition $\frac{\partial u}{\partial x}|_{x=0}=g$ is specified, the SAT term $\boldsymbol{b}^{SAT}$ (a vector) takes the following form:

$$\boldsymbol{b}^{SAT}=\alpha(\boldsymbol{B}\boldsymbol{u}-g)\boldsymbol{e}_0. \tag{IV.6}$$

Here, $\boldsymbol{u}$ is the grid vector (the numerical approximation to the solution) for boundary data g. $\boldsymbol{B}=\boldsymbol{e}_0^T\boldsymbol{D}_x$ is the operator that computes a one-sided difference approximation at the domain boundary. α is a penalty parameter that is chosen under stability constraints from a discrete energy estimate. More detailed examples of SAT terms in practice can be found in [28]. Multiple boundary conditions can be enforced by adding the corresponding $\boldsymbol{b}^{SAT}$ terms to the discretization of the PDE. Compared to the traditional method of using injection or strong enforcement of boundary/interface conditions that would destroy the SBP property defined in equations IV.2 and IV.4, using SAT terms enables proof of the method's stability [63].

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

We use the following example from [73] to showcase an example of applying the SBP-SAT method for PDEs. Let's consider the advection problem in 1D.

$$\begin{aligned}\boldsymbol{u}_t+\boldsymbol{u}_x&=0,\quad 0<x<1,t>0\\ \boldsymbol{u}(0,t)&=\boldsymbol{g}(t),\quad t>0\\ \boldsymbol{u}(x,t)&=\boldsymbol{h}(x),\quad 0<x<1\end{aligned} \tag{IV.7}$$

where both g and h are known for initial and boundary conditions. The problem Equation IV.7 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 subsection IV-B. By applying SBP-SAT discretization in space to Equation IV.7, we get

$$\begin{aligned}\boldsymbol{u}_t+D_1\boldsymbol{u}&=P^{-1}\sigma(\boldsymbol{u}_0-\boldsymbol{g})\boldsymbol{e}_0,\quad t>0\\ \boldsymbol{u}(0)&=\boldsymbol{h}\end{aligned} \tag{IV.8}$$

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 IV.8 is strongly stable, we have

$$||\boldsymbol{u}(t)||^2\le K(t)(||\boldsymbol{h}||^2+\max_{\tau\in[0,t]}|\boldsymbol{g}(\tau)|^2) \tag{IV.9}$$

The $K(t)$ in Equation IV.9 is independent of the data and bounded for any finite t and meshsize Δx. Further details about $K(t)$ are given in [85, 36]. Applying the energy method

by multiplying the equation Equation IV.8 with $\boldsymbol{u}^T P$ and adding the transpose with the SBP property Equation IV.2, we find

$$\frac{d}{dt}||\boldsymbol{u}||_P^2 = -\frac{\sigma^2}{1+2\sigma}\boldsymbol{g}^2 - \boldsymbol{u}_N^2 + \frac{[(1+2\sigma)\boldsymbol{u}_0 - \sigma\boldsymbol{g}]^2}{1+2\sigma} \quad \text{(IV.10)}$$

By time-integration, this leads to an estimate of the form Equation IV.9 for $\sigma < -1/2$.

F. 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

$$-\Delta u = f, \quad \text{for } (x,y) \in \Omega, \quad \text{(IV.11a)}$$
$$u = g_{\mathrm{W}}, \quad x = 0, \quad \text{(IV.11b)}$$
$$u = g_{\mathrm{E}}, \quad x = 1, \quad \text{(IV.11c)}$$
$$\boldsymbol{n}\cdot\nabla u = g_{\mathrm{S}}, \quad y = 0, \quad \text{(IV.11d)}$$
$$\boldsymbol{n}\cdot\nabla u = g_{\mathrm{N}}, \quad y = 1, \quad \text{(IV.11e)}$$

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 (IV.11) is given by

$$-\boldsymbol{D}_2\boldsymbol{u} = \boldsymbol{f} + \boldsymbol{b}^N + \boldsymbol{b}^S + \boldsymbol{b}^W + \boldsymbol{b}^E, \quad \text{(IV.12)}$$

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

$$\boldsymbol{b}^W = \alpha\left(\boldsymbol{H}^{-1}\otimes\boldsymbol{I}\right)(\boldsymbol{E}_{\mathrm{W}}\boldsymbol{u} - \boldsymbol{e}_{\mathrm{W}}^T\boldsymbol{g}_{\mathrm{W}}) \quad \text{(IV.13)}$$
$$-\left(\boldsymbol{H}^{-1}\boldsymbol{e_0}\boldsymbol{d_0}^T\otimes\boldsymbol{I}\right)(\boldsymbol{E}_{\mathrm{W}}\boldsymbol{u} - \boldsymbol{e}_{\mathrm{W}}^T\boldsymbol{g}_{\mathrm{W}}), \quad \text{(IV.14)}$$

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 [28]. System (IV.12) can be rendered SPD by multiplying from the left by $(\boldsymbol{H}\otimes\boldsymbol{H})$, producing the sparse linear system $\boldsymbol{A}\boldsymbol{u} = \boldsymbol{b}$.

V. Iterative Methods

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

A. Stationary Iterative Methods

Stationary iterative methods can be expressed in the simple form

$$\boldsymbol{u}^{k+1} = \boldsymbol{Q}\boldsymbol{u}^k + \boldsymbol{q} \quad \text{(V.1)}$$

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 [74]

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

$$\boldsymbol{A} = \boldsymbol{S} + (\boldsymbol{A} - \boldsymbol{S}) \quad \text{(V.2)}$$

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

$$\boldsymbol{S}\boldsymbol{u} = (\boldsymbol{S} - \boldsymbol{A})\boldsymbol{u} + \boldsymbol{f} \quad \text{(V.3)}$$

and the iterative scheme of the splitting method is defined as

$$\boldsymbol{u}^{k+1} = \boldsymbol{S}^{-1}((\boldsymbol{S} - \boldsymbol{A})\boldsymbol{u}^k + f) \quad \text{(V.4)}$$

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

$$\boldsymbol{M} = \boldsymbol{S}^{-1}(\boldsymbol{S} - \boldsymbol{A}) \quad \text{(V.5)}$$

.

If we define $\boldsymbol{Q} = \boldsymbol{M}$ and $\boldsymbol{q} = \boldsymbol{S}^{-1}\boldsymbol{f}$, then Equation V.4 can be written as $\boldsymbol{u} = \boldsymbol{Q}\boldsymbol{u} + \boldsymbol{q}$, and it satisfy

$$\boldsymbol{e}^{k+1} = \boldsymbol{M}\boldsymbol{e}^k \quad \text{(V.6)}$$

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

$$\rho(\boldsymbol{M}) < 1 \quad \text{(V.7)}$$

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.

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

$$\boldsymbol{A} = \boldsymbol{D} + (\boldsymbol{A} - \boldsymbol{D}) \text{ or } \boldsymbol{A} = \boldsymbol{D} + (-\boldsymbol{L} - \boldsymbol{U}) \quad \text{(V.8)}$$

Where $-\boldsymbol{L}$ denotes the strictly lower triangle and $-\boldsymbol{U}$ denotes the strictly upper triangle of the matrix $\boldsymbol{A}$. Similar to Equation V.4, the Jacobi method is then

$$\boldsymbol{u}^{k+1} = \boldsymbol{D}^{-1}((\boldsymbol{L} + \boldsymbol{U})\boldsymbol{u}^k + \boldsymbol{f}) \quad \text{(V.9)}$$

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

$$u_i^{k+1} = \frac{1}{A_{ii}}(f_i - \sum_{j=1, i\neq j} A_{ij} u_j^k) \tag{V.10}$$

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

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

$$\boldsymbol{A} = (\boldsymbol{D} - \boldsymbol{L}) + (\boldsymbol{A} - (\boldsymbol{D} - \boldsymbol{L})) \text{ or } \boldsymbol{A} = \boldsymbol{D} - \boldsymbol{L} + (-\boldsymbol{U}) \tag{V.11}$$

Similar to Equation V.4, the Gauss-Seidel method is then

$$\boldsymbol{u}^{k+1} = (\boldsymbol{D} - \boldsymbol{L})^{-1}(\boldsymbol{U}\boldsymbol{u}^k + \boldsymbol{f}) \tag{V.12}$$

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

$$\boldsymbol{D}\boldsymbol{u}^{k+1} - \boldsymbol{L}\boldsymbol{u}^{k+1} = \boldsymbol{U}\boldsymbol{u}^k + \boldsymbol{f} \tag{V.13}$$

and

$$\boldsymbol{u}^{k+1} = \boldsymbol{D}^{-1}(\boldsymbol{L}\boldsymbol{u}^{k+1} + \boldsymbol{U}\boldsymbol{u}^k + \boldsymbol{f}) \tag{V.14}$$

and the index form is given as

$$u_i^{k+1} = \frac{1}{A_{ii}}(f_i - \sum_{j=1}^{i-1} A_{ij} u_j^{k+1} - \sum_{i+1}^{n} A_{ij} u_j^k) \tag{V.15}$$

3) Comparison of the Jacobi and the Gauss-Seidel Method: It might seem that in Equation V.14 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 V.15

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 $j = 1 \dots (i-1)$, 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 [38], 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 [35]

$$|A_{ii}| > \sum_{j=1, j\neq i}^{n} |A_{ij}| \text{ for all } i \tag{V.16}$$

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 V.16, 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. [11] 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.

4) Relaxation methods: Relaxation methods are also stationary iterative methods, thus they can also be presented in the form Equation V.1. 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

$$u_i^{k+1} := (1-\omega)u_i^k + \omega \breve{u}_i^{k+1} \tag{V.17}$$

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 [35] and [96].

B. 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 [54]. The Krylov subspace takes the form

$$\mathcal{K}_k(A, \boldsymbol{v}) := \text{span}\{\boldsymbol{v}, A\boldsymbol{v}, \ldots, A^{k-1}\boldsymbol{v}\} \tag{V.18}$$

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

1) Conjugate Gradient Method: The conjugate gradient (CG) method was developed by *Hestenes & Stiefel* [41] 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 $F(x) = min$

$$F(\boldsymbol{u}) = \frac{1}{2}\boldsymbol{u}^T\boldsymbol{A}\boldsymbol{u} - \boldsymbol{f}^T\boldsymbol{u} \tag{V.19}$$

is equivalent to getting its derivative

$$\text{grad}F(\boldsymbol{u}) = \boldsymbol{A}\boldsymbol{u} - \boldsymbol{f} \tag{V.20}$$

equal to the zero vector

$$\text{grad}F(\boldsymbol{u}) = 0 \tag{V.21}$$

The CG method is an iterative minimizer of the given quadratic form and therefore an iterative solver for the linear equation system $\boldsymbol{Au = f}$ 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

$$F(\boldsymbol{u}^k + \lambda\boldsymbol{p}^k) = min \tag{V.22}$$

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

$$\begin{aligned}(\frac{1}{2}\boldsymbol{p}^{k^T}\boldsymbol{A}\boldsymbol{p}^k)\lambda^2 + (\boldsymbol{p}^{k^T}\boldsymbol{A}\boldsymbol{u}^k - \boldsymbol{p}^{k^T}\boldsymbol{f})\lambda \\ + ((\frac{1}{2}\boldsymbol{u}^{k^T}\boldsymbol{A}\boldsymbol{u}^k) - \boldsymbol{u}^{k^T}) = min\end{aligned} \tag{V.23}$$

This quadratic form is minimized for

$$\lambda = \frac{\boldsymbol{p}^{k^T}(\boldsymbol{f} - \boldsymbol{A}\boldsymbol{u}^k)}{\boldsymbol{p}^{k^T}\boldsymbol{A}\boldsymbol{p}^k} \tag{V.24}$$

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

$$-\text{grad}F(u^k) = \boldsymbol{f} - \boldsymbol{A}\boldsymbol{u}^k = \boldsymbol{r}^k \tag{V.25}$$

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

$$\lambda_k = \frac{\boldsymbol{r}^{k^T}\boldsymbol{r}^k}{\boldsymbol{r}^{k^T}\boldsymbol{A}\boldsymbol{r}^k} \tag{V.26}$$

$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \lambda_k\boldsymbol{r}^k \tag{V.27}$$

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

$$A\boldsymbol{p}^{k+1} \perp \boldsymbol{p}^k \tag{V.28}$$

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

$$\begin{aligned}\boldsymbol{r}^0 &= \boldsymbol{f} - \boldsymbol{A}\boldsymbol{u}^0 \\ \boldsymbol{p}^0 &= \boldsymbol{r}^0\end{aligned} \tag{V.29}$$

The following equations describe the algorithm of the conjugate gradient method

$$\lambda_k = \frac{\boldsymbol{r}^{k^T}\boldsymbol{p}^k}{\boldsymbol{p}^{k^T}\boldsymbol{A}\boldsymbol{p}^k} \tag{V.30}$$

$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \lambda_k\boldsymbol{p}^k \tag{V.31}$$

$$\boldsymbol{r}^{k+1} = \boldsymbol{r}^k - \lambda_k\boldsymbol{A}\boldsymbol{p}^k \tag{V.32}$$

$$\boldsymbol{p}^{k+1} = \boldsymbol{r}^k - \frac{\boldsymbol{r}^{k+1^T}\boldsymbol{A}\boldsymbol{p}^k}{\boldsymbol{p}^{k^T}\boldsymbol{A}\boldsymbol{p}^k}\boldsymbol{p}^k \tag{V.33}$$

As shown in [35], for an efficient implementation, it is possible to use an alternative form for λ_k and p^{k+1}

$$\lambda_k = \frac{\boldsymbol{r}^{k^T}\boldsymbol{r}^k}{\boldsymbol{p}^{k^T}\boldsymbol{A}\boldsymbol{p}^k} \tag{V.34}$$

$$\boldsymbol{p}^{k+1} = \boldsymbol{r}^k + \frac{\boldsymbol{r}^{k+1^T}\boldsymbol{r}^k}{\boldsymbol{r}^{k^T}\boldsymbol{r}^k}\boldsymbol{p}^k \tag{V.35}$$

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.

VI. Multigrid Methods

A. General 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 [29] [87]. In practice, it can be implemented as a standalone method or as a preconditioner for other iterative methods such as the conjugate gradient method [86]. Various multigrid methods are used in different branches of applied mathematics and engineering. such as electromagnetics [81] and fluid dynamics [2].

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 [17] 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:

$$Lu = f, \text{ in } \Omega \tag{VI.1}$$
$$Hu = g, \text{ on } \partial\Omega \tag{VI.2}$$

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.

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) Fine-grid discretization: Consider a *fine grid* meshing Ω_1 on Ω. A discrete linear system associated to Equation VI.1 on this fine grid Ω_1 has the general form

$$L_1\mathbf{u} = \mathbf{F} \tag{VI.3}$$

where L_1 is the discrete version of the operator L in Equation VI.1 which also include boundary conditions in Equation VI.2. The vector $\mathbf{F}$ approximates f on the grid points of Ω_1 which already incorporates data for the boundary condition**g** in Equation VI.2. $\mathbf{u}$ 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.

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 VI.3 is through solving

$$\mathbf{w}_\tau + L_1\mathbf{w}(\tau) = \mathbf{F},\ 0 < \tau < \Delta\tau \tag{VI.4}$$
$$\mathbf{w}(0) = \mathbf{u}^{(0)} \tag{VI.5}$$

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

$$\mathbf{w}(\Delta\tau) = e^{-L_1\Delta\tau}\mathbf{u}^0 + (I_1 - e^{-L_1\Delta\tau})L_1^{-1}\mathbf{F} \tag{VI.6}$$

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

$$||\mathbf{w}(\Delta\tau) - \mathbf{u}|| < ||\mathbf{u}^{(0)} - \mathbf{u}|| \tag{VI.7}$$

Smoothing technique for the solution can be defined as follows

$$\begin{aligned}&\mathbf{w}^k = S\mathbf{w}^{k-1} + (I_1 - S)L_1^{-1}\mathbf{F}, k = 1, \dots \nu\\ &\mathbf{w}^0 = \mathbf{u}^0\end{aligned} \tag{VI.8}$$

where S is the smoother. If S is an exponential smoother $S_{\text{exp}} = e^{-L_1\Delta\tau}$, this will yield the pseudo time-marching procedure in Equation VI.4. This iterative method would converge after ν steps to

$$\mathbf{w} = S^\nu\mathbf{u}^0 + (I_1 - S^\nu)L_1^{-1}\mathbf{F} \tag{VI.9}$$

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

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

$$L_1\mathbf{e} = \mathbf{F} - L_1\mathbf{w} \tag{VI.10}$$

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

$$L_2\mathbf{d} = I_r(\mathbf{F} - L_1\mathbf{w}) \tag{VI.11}$$

This problem is obtained from the finer grid problem Equation VI.10 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

$$L_2 = I_rL_1I_p \tag{VI.12}$$

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 VI.12 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

$$(I_p\mathbf{v})_m = \begin{cases} v_j, & m = 2j, j = 0, \ldots, N/2 \\ \frac{1}{2}(v_j + v_{j+1}), & m = 2j+1, j = 0, \ldots, N/2 \end{cases} \tag{VI.13}$$

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

$$I_r = I_p^T/C \tag{VI.14}$$

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.

5) Fine-grid update: Finally, we update the fine grid solution with correction $\mathbf{d}$ as

$$\mathbf{u}^{(1)} = \mathbf{w} + I_p\mathbf{d} \tag{VI.15}$$

The relation Equation VI.15, together with Equation VI.9 and Equation VI.11 provides an iterative method for solving the steady-state problem

$$\mathbf{u}^{n+1} = M\mathbf{u}^n + N\mathbf{F} \tag{VI.16}$$

where

$$M = CS^\nu \tag{VI.17}$$

$$C = I_1 - I_p L_2^{-1} I_r L_1 \tag{VI.18}$$

$$N = (I_1 - M)L_1^{-1} \tag{VI.19}$$

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

$$\mathbf{e}^{(n+1)} = M\mathbf{e}^{(n)} \tag{VI.20}$$

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.

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.

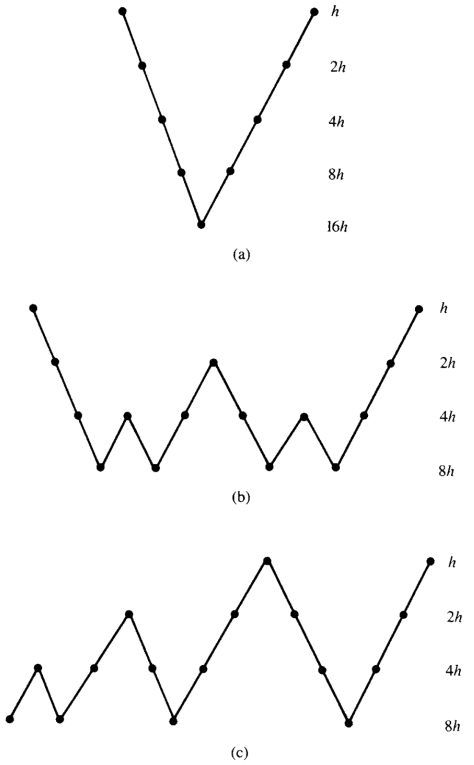

Fig. 1: Schedule of grids for (a) V-cycle, (b) W-cycle, and (c) FMG scheme, all on four levels. [16]

B. Algebraic Multigrid

The classical multigrid formed around the geometric structure has been generalized that the multigrid is analyzed in terms of the matrix properties [64]. 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 [15, 59, 71]. A detailed overview of the algebraic multigrid can be found in this recent paper [93]. 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:*

$$\mathbf{R}^m = \mathcal{R}(A) \oplus \mathcal{N}(A^T) \tag{VI.21}$$

$$\mathbf{R}^n = \mathcal{R}(A) \oplus \mathcal{N}(A) \tag{VI.22}$$

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[16].

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 [16].

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 [88]. 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` [71]. A parallel implementation of the CF-based AMG can be found in the BoomerAMG package in the Hypre library [95]. The Trilinos package includes `ML` as a parallel SA-based AMG solver [32]. Finally, PyAMG includes a number of AMG variants for testings, and Cusp distributes with a standard SA implementation for use on a GPU [25, 13].

C. 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 VI.12 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 [72, 73].

To overcome this issue, consider defining the restriction operator as

$$I_r = H_2^{-1} I_p^T H_1 \tag{VI.23}$$

which was first introduced in [73]. This involves the coarse grid SBP norm H_2 and is obtained by enforcing that two scalar products

$$(\phi_1, \psi_1)_{H_1} = (\phi_1 H_1 \psi_1) \tag{VI.24}$$
$$(\phi_2, \psi_2)_{H_2} = (\phi_2 H_2 \psi_2) \tag{VI.25}$$

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 IV-A. 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 [38].

$$(I_p, \xi_2, \xi_1)_{H_1} = (\xi_2, I_r \xi_1)_{H_2} \tag{VI.26}$$

by using Equation VI.23, it is possible to build pairs of consistent and accurate prolongation and restriction operators.

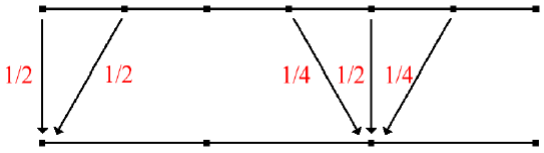

Fig. 2: The 2nd-order SBP-preserving restriction operator I_r

The following definition of the SBP-preserving interpolation operators was given in [73], 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 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 [83].

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.

1) SBP-preserving interpolation applied to the first derivative: Using Galerkin condition Equation VI.12 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

$$D_{1,2} = I_r D_{1,1} I_p \tag{VI.27}$$

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

$$(\phi, D_1 \psi)_H = \phi_N \psi_N - \phi_0 \psi_0 - (D_1 \phi, \psi)_H \tag{VI.28}$$

with the adjoint relation Equation VI.26 as follows

$$\begin{aligned}(\phi_2, D_{1,2}\psi_2)_{H_2} &= (\phi_2, I_r(D_{1,1} I_p \phi_2))_{H_2} \\ &= (I_p \phi_2, D_{1,1}(I_p \psi_2))_{H_1}\end{aligned} \tag{VI.29}$$

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

$$\begin{aligned}(\phi_2, D_{1,2}\psi_2)_{H_2} &= (I_p, \phi_2)_N (I_p, \psi_2)_N \\ &\quad - (I_p, \phi_2)_0 (I_p, \psi_2)_0 - (D_{1,1}(I_p \phi_2), I_p \psi_2)_{H_1}\end{aligned} \tag{VI.30}$$

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 VI.26 to the right-hand side of Equation VI.30, we obtain

$$(\phi_2, D_{1,2}\psi_2)_{H_2} = \phi_{2,N/2}\psi_{2,N/2} - \phi_{2,0}\psi_{2,0} - (D_{1,2}\phi_2, \psi_2)_{H_2} \tag{VI.31}$$

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 [72].

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 VI.23 lead to a coarse-grid second derivative operator $D_{2,2}$ which preserves the summation-by-parts property Equation IV.4. We can show that by rewriting the left hand side of the Equation IV.4 for $D_{2,2}$ and the two coarse-grid functions ϕ_2 and ψ_2 by using Equation VI.26.

$$(\phi_2, D_{2,2}\psi_2)_{H_2} = (\phi_2, I_r, D_{2,1}I_p\psi_2)_{H_2} = (I_p\phi_2, D_{2,1}I_p\psi_2)_{H_1} \tag{VI.32}$$

By applying the SBP property Equation IV.4 for the fine-grid second derivative $D_{2,1}$, we have

$$\begin{aligned}(\phi_2, D_{2,2}\psi_2)_{H_2} &= (I_p\phi_2)_N(SI_p\psi_2)_N \\ &\quad - (I_p\phi_2)_0(SI_p\psi_2)_0 - (SI_p\phi_2)^T A(SI_p\psi_2)_{H_2}\end{aligned} \tag{VI.33}$$

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

$$\begin{aligned}(\phi_2, D_{2,2}\psi_2)_{H_2} &= \phi_{2,N/2}(S\phi_2)_{N/2} \\ &\quad - \phi_{2,0}(S\phi_2)_0 - (SI_p\phi_2)^T A(SI_p\psi_2)_{H_2}\end{aligned} \tag{VI.34}$$

where S is equivalent to $\boldsymbol{d}_0^T$ in subsection IV-A which approximates the first derivative at the boundaries.

Additional proofs or propositions to SBP-preserving interpolations can also be found in [72]. 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 [72].

VII. Preconditioner

A. Preconditioner for Linear Systems

As we discussed stationary iterative methods in subsection V-A, we now review these methods from a preconditioning perspective.

The Jacobi and Gauss-Seidel iterations are of the form

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

in which

$$\boldsymbol{Q}_{JA} = \boldsymbol{I} - \boldsymbol{D}^{-1}\boldsymbol{A} \tag{VII.2}$$

$$\boldsymbol{Q}_{GS} = \boldsymbol{I} - \boldsymbol{D} - \boldsymbol{L}^{-1}\boldsymbol{A} \tag{VII.3}$$

for the Jacobi and Gauss-Seidel iterations, respectively. Given the matrix splitting

$$\boldsymbol{A} = \boldsymbol{S} - (\boldsymbol{S} - \boldsymbol{A}) \tag{VII.4}$$

a *linear fixed-point iteration* can be defined by the recurrence

$$\boldsymbol{u}^{k+1} = \boldsymbol{S}^{-1}(\boldsymbol{S} - \boldsymbol{A})\boldsymbol{u}^k + \boldsymbol{S}^{-1}\boldsymbol{f} \tag{VII.5}$$

which has the form Equation VII.1 with

$$\boldsymbol{Q} = \boldsymbol{S}^{-1}(\boldsymbol{S} - \boldsymbol{A}) = \boldsymbol{I} - \boldsymbol{S}^{-1}\boldsymbol{A}, \quad \boldsymbol{q} = \boldsymbol{S}^{-1}\boldsymbol{f} \tag{VII.6}$$

For example, for the Jacobi iteration, $\boldsymbol{S} = \boldsymbol{D}, \boldsymbol{S} - \boldsymbol{A} = \boldsymbol{D} - \boldsymbol{A}$, while for the Gauss-Seidel iteration $\boldsymbol{S} = \boldsymbol{D} - \boldsymbol{L}, \boldsymbol{S} - \boldsymbol{A} = \boldsymbol{D} - \boldsymbol{L} - \boldsymbol{A} = \boldsymbol{U}$.

The iteration $\boldsymbol{u}^{k+1} = \boldsymbol{Q}\boldsymbol{u}^k + \boldsymbol{q}$ can also be viewed as a technique for solving the system

$$(\boldsymbol{I} - \boldsymbol{Q})\boldsymbol{u} = \boldsymbol{q} \tag{VII.7}$$

Since $\boldsymbol{Q}$ has the form $\boldsymbol{Q} = \boldsymbol{I} - \boldsymbol{S}^{-1}\boldsymbol{A}$, this system can be rewritten as

$$\boldsymbol{S}^{-1}\boldsymbol{A}\boldsymbol{u} = \boldsymbol{S}^{-1}\boldsymbol{f} \tag{VII.8}$$

We call this system that has the same solution as the original system $\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f}$ the *preconditioned system* and $\boldsymbol{S}$ the preconditioning matrix or preconditioner. It's often to use $\boldsymbol{M}$ to denote $\boldsymbol{S}$ when the splitting matrix $\boldsymbol{S}$ is used in the preconditioning. In other words, a *relaxation scheme* is equivalent to a *fixed-point iteration* on a *preconditioned system*. The preconditioning matrices can be easily derived for the Jacobi, Gauss-Seidel, SOR and SSOR iterations as follows

$$\boldsymbol{M}_{JA} = \boldsymbol{D} \tag{VII.9}$$

$$\boldsymbol{M}_{GS} = \boldsymbol{D} - \boldsymbol{L} \tag{VII.10}$$

$$\boldsymbol{M}_{SOR} = \frac{1}{\omega}(\boldsymbol{D} - \omega\boldsymbol{L}) \tag{VII.11}$$

$$\boldsymbol{M}_{SSOR} = \frac{1}{\omega(2-\omega)}(\boldsymbol{D} - \omega\boldsymbol{L})\boldsymbol{D}^{-1}(\boldsymbol{D} - \omega\boldsymbol{U}) \tag{VII.12}$$

The matrix $\boldsymbol{M}^{-1}$ should be symmetric and positive definite. Even though $\boldsymbol{M}$ is sparse most of the time, there is no guarantee that the $\boldsymbol{M}^{-1}$ is sparse. And this limits the number of techniques that can be applied to solve the preconditioned system. Also the computation of $\boldsymbol{M}^{-1}\boldsymbol{b}$ for any vector $\boldsymbol{b}$ so the actual solution to the problem can be easily obtained from the solution to the preconditioned system.

B. General Convergence Results

In this section, we examine the convergence behaviors of the general preconditioners above. The detailed analysis can be found in [75]. We choose the most important results to present here with notations adapted to match the other sections of the paper. All methods seen in the previous section define a sequence of iterates of the form

$$\boldsymbol{u}^{k+1} = \boldsymbol{Q}\boldsymbol{u}^k + \boldsymbol{q} \tag{VII.13}$$

where $\boldsymbol{Q}$ is the iteration matrix. We need to answer these two questions

- If the iteration converges, is the limit indeed a solution of the original system?
- Under which conditions does the iteration converge?
- How fast is the convergence?

If the above iteration converges to $\boldsymbol{u}$, and it satisfies

$$\boldsymbol{u} = \boldsymbol{Q}\boldsymbol{u} + \boldsymbol{q} \tag{VII.14}$$

Recall the definition in Equation VII.6, it's easy to verify the $\boldsymbol{u}$ satisfies $\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f}$. This answers the first question. We now consider the next two questions.

If $\boldsymbol{I} - \boldsymbol{Q}$ is nonsingular, then there is a solution $\boldsymbol{u}^*$ to the equation Equation VII.14. Substracting Equation VII.14 from Equation VII.13 yields

$$\boldsymbol{u}^{k+1} - \boldsymbol{u}^* = \boldsymbol{Q}(\boldsymbol{u}^k - \boldsymbol{u}^*) = \cdots \boldsymbol{Q}^{k+1}(\boldsymbol{u}^0 - \boldsymbol{u}^*) \tag{VII.15}$$

If the spectral radius of the iteration matrix $\boldsymbol{Q}$ is less than 1, then $\boldsymbol{u}^k - \boldsymbol{u}^0$ converges to zero. And the iteration Equation VII.13 converges toward the solution defined by Equation VII.14. Conversely, the relation

$$\boldsymbol{u}^{k+1} - \boldsymbol{u}^k = \boldsymbol{Q}(\boldsymbol{u}^k - \boldsymbol{u}^{k-1}) = \cdots \boldsymbol{Q}^k(\boldsymbol{q} - (\boldsymbol{I} - \boldsymbol{Q})\boldsymbol{u}^0)) \tag{VII.16}$$

shows that if the iteration converges for any $\boldsymbol{u}^0$ and $\boldsymbol{q}$, then $\boldsymbol{Q}^k\boldsymbol{b}$ converges to zero for any vector $\boldsymbol{b}$. As a result, $\rho(\boldsymbol{Q})$ must be less than 1. The following theorem is proved:

Theorem 2. *Let $\boldsymbol{Q}$ be a square matrix such that $\boldsymbol{Q} < 1$. Then $\boldsymbol{I} - \boldsymbol{Q}$ is non-singular and iteration Equation VII.13 converges for any $\boldsymbol{q}$ and $\boldsymbol{u}^0$. Conversely, if the iteration Equation VII.13 converges for any $\boldsymbol{q}$ and $\boldsymbol{u}^0$, then $\rho(\boldsymbol{Q}) < 1$*

Since it is often expensive to compute the spectral radius of a matrix, sufficient conditions that guarantee convergence can be useful in practice. One such sufficient condition could be obtained by utilizing the inequality $\rho(\boldsymbol{Q} \leq ||\boldsymbol{Q}||)$ for any matrix norm.

Corollary 2.1. *Let $\boldsymbol{Q}$ be a square matrix such that $||\boldsymbol{Q}|| < 1$ for some matrix norm $||\cdot||$.. Then $\boldsymbol{I} - \boldsymbol{Q}$ is non-singular and the iteration Equation VII.13 converges for any initial vector $\boldsymbol{u}^0$*

Other than knowing that Equation VII.13 converges, we can also know how fast it converges. The error $\boldsymbol{e}^k = \boldsymbol{u}^k - \boldsymbol{u}^*$ at step k satisfies that

$$\boldsymbol{e}^k = \boldsymbol{Q}^k\boldsymbol{e}^0 \tag{VII.17}$$

We can prove this by expressing $\boldsymbol{Q}$ in Jordan canonical form

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{J}\boldsymbol{X}^{-1} \tag{VII.18}$$

For simplicity, we assume that there is only one eigenvalue of $\boldsymbol{Q}$ of the largest modulus denoted by λ. Then

$$\boldsymbol{e}^k = \lambda^k\boldsymbol{X}(\frac{\boldsymbol{J}}{\lambda})^k\boldsymbol{X}^{-1}\boldsymbol{e}^0 \tag{VII.19}$$

The matrix $\boldsymbol{J}/\lambda$ shows that all its blocks, except the block associated with the eigenvalue λ, converge to zero as $k \to \infty$. Let this Jordan block be of size p, and of the form $\boldsymbol{J}_\lambda = \lambda\boldsymbol{I} + \boldsymbol{L}$. Here $\boldsymbol{L}$ is nilpotent of p, i.e. $\boldsymbol{L}^p = 0$. Then for $k \geq p$

$$\boldsymbol{J}_\lambda^k = (\lambda\boldsymbol{I} + \boldsymbol{L})^k = \lambda^k(\boldsymbol{I} + \lambda^{-1}\boldsymbol{L})^k = \lambda^k(\sum_{i=0}^{p-1}\lambda^{-i}\binom{k}{i}\boldsymbol{L}^i) \tag{VII.20}$$

If k is large enough, then for any λ the dominant term in the above sum is the last term, i.e.,

$$\boldsymbol{J}_\lambda^k \approx \lambda^{k-p+1}\binom{k}{p-1}\boldsymbol{L}^{p-1} \tag{VII.21}$$

Thus, the norm of $\boldsymbol{e}^k = \boldsymbol{Q}^k\boldsymbol{e}^0$ has the asymptotical form

$$||\boldsymbol{e}^k|| \approx C \times |\lambda^{k-p+1}|\binom{k}{p-1} \tag{VII.22}$$

where C is some constant. The *convergence factor* of a sequence is the limit

$$\rho = \lim_{k\to\infty}(\frac{||\boldsymbol{e}^k||}{||\boldsymbol{e}^0||})^{1/k} \tag{VII.23}$$

From above analysis, $\rho = \rho(\boldsymbol{Q})$. The *convergence rate* τ is the natural logarithm of the inverse of the convergence factor

$$\tau = -\ln\rho \tag{VII.24}$$

Note that the above definition depends on the initial vector $\boldsymbol{e}^0$, so it maybe be termed a *specific* convergence factor. A *general* convergence factor can be defined by

$$\phi = \lim_{k\to\infty}(\max_{\boldsymbol{u}^0\in\mathbb{R}^n}\frac{||\boldsymbol{e}^k||}{||\boldsymbol{e}^0||})^{1/k} \tag{VII.25}$$

This satisfies

$$\phi = \lim_{k\to\infty}(\max_{\boldsymbol{u}^0\in\mathbb{R}^n}\frac{||\boldsymbol{Q}^k\boldsymbol{e}^0||}{||\boldsymbol{e}^0||})^{1/k} \tag{VII.26}$$

$$= \lim_{k\to\infty}(||\boldsymbol{Q}^k||)^{1/k} = \rho(\boldsymbol{Q}) \tag{VII.27}$$

Thus, the global asymptotic convergence factor is equal to the spectral radius of the iteration matrix $\boldsymbol{G}$. The general convergence rate differs from the specific rate only when the initial error does not have any components in the invariant subspace associated with the dominant eigenvalues. Since it is hard to know a priori, the general convergence factor is more useful in practice.

Using convergence analysis here, the convergence criteria for several iterative methods such as Richardson's Iteration, and regular splitting can be derived with simpler forms. One type of matrices that is worth notice is diagonally dominant matrices. We begin with a few standard definitions

Definition 4. *A matrix A is*

- *(weakly) diagonally dominant if*

$$|a_{j,j}| \geq \sum_{i=1, i\neq j}^{i=n} |a_{i,j}|, \quad j = 1, \ldots, n \tag{VII.28}$$

- *strictly diagonally dominant if A is irreducible, and*

$$|a_{j,j}| > \sum_{i=1, i\neq j}^{i=n} |a_{i,j}|, \quad j = 1, \ldots, n \tag{VII.29}$$

- *irreducibly diagonally dominant if*

$$|a_{j,j}| \geq \sum_{i=1, i\neq j}^{i=n} |a_{i,j}|, \quad j = 1, \ldots, n \tag{VII.30}$$

with strict inequality for at least one j

The diagonally dominant matrices are important as many matrices from the discretization of PDEs are diagonally dominant. When solving these linear systems with iterative methods, the spectral radius can be estimated using Gershgorin's theorem. Gershgorin's theorem allows rough locations for all eigenvalues of $\boldsymbol{A}$ to be determined. In situations where $\boldsymbol{A}$ is so large that the eigenvalues of $\boldsymbol{A}$ are unable to obtain, for example, a linear system from extremely fine discretization, the spectral radius can be directly obtained via the entries of the matrix $\boldsymbol{A}$. The simplest such result is the bound

$$|\lambda_i| \leq ||\boldsymbol{A}|| \tag{VII.31}$$

for any matrix norm. Gershgorin's theorem provides a more precise localization result

Theorem 3 (Gershgorin)**.** *Any eigenvalue λ of a matrix $\boldsymbol{A}$ is located in one of the closed discs of the complex plane centered at $a_{i,i}$ and has the radius*

$$\rho_i = \sum_{j=1, j\neq i}^{j=n} |a_{i,j}| \tag{VII.32}$$

In other words,

$$\forall \lambda \in \sigma(\boldsymbol{A}), \exists i \text{ such that} |\lambda - a_{i,i}| \leq \sum_{j=1, j\neq i}^{j=n} |a_{i,j}| \tag{VII.33}$$

Proof. Let x be an eigenvector associated with an eigenvalue λ, and let m be an index of the component of largest modulus in $\boldsymbol{x}$, scale $\boldsymbol{x}$ so that $|\xi_m| = 1$, and $\xi_i \leq 1, \forall i \neq m$. Since $\boldsymbol{x}$ is an eigenvector, then

$$(\lambda - a_{m,m})\xi_m = -\sum_{j=1, j\neq m}^{n} a_{m,j}\xi_j \tag{VII.34}$$

which gives

$$|\lambda - a_{m,m}| \leq \sum_{j=1, j\neq m}^{n} |a_{m,j}||\xi_j| \leq \sum_{j=1, j\neq m}^{n} |a_{m,j}| = \rho_m \tag{VII.35}$$

□

This result also holds for the transpose of $\boldsymbol{A}$, so this theorem can also be formulated based on column sums instead of row sums. The n discs defined in the theorem are called Gershgorin discs. The theorem states that the union of these n discs contains the spectrum of $\boldsymbol{A}$. It can also be shown that if there are m Gershgorin discs whose union S is disjoint from all other discs, then S contains exactly m eigenvalues (with multiplicities counted). Additional refinement which has important consequences concerns of a particular case when $\boldsymbol{A}$ is irreducible is given here.

Theorem 4. *Let $\boldsymbol{A}$ be an irreducible matrix and assume that an eigenvalue λ of $\boldsymbol{A}$ lies on the boundary of the union of the n Gershgorin discs. Then λ lies on the boundary of all Gershgorin discs*

This theorem and its proof can be found in [75] where an immediate corollary of the Gershgorin theorem and this theorem follows

Corollary 4.1. *If a matrix $\boldsymbol{A}$ is strictly diagonally dominant or irreducibly diagonally dominant, then it is nonsingular.*

This leads to the following theorem

Theorem 5. *If $\boldsymbol{A}$ is a strictly diagonally dominant or an irreducibly diagonally dominant matrix, then the associated Jacobi and Gauss-Seidel iterations converge for any $\boldsymbol{u}^0$.*

The convergence condition for SPD matrices is given as follows

Theorem 6. *if $\boldsymbol{A}$ is symmetric with positive diagonal elements and for $0 < \omega < 2$, SOR converges for any $\boldsymbol{u}^0$ if and only if $\boldsymbol{A}$ is positive definite.*

These theorems for convergence play an important role in the study of various preconditioners. That's why we dedicate a huge portion of the paper to them. More iterative methods and preconditioners as well as their convergence criteria can be found in [75].

C. Multigrid Preconditioned Conjugate Gradient

In the previous sections, we introduce the classical iterative solvers and Krylov subspace methods as a solver. Moreover, we show that the classical iterative solvers can be used in the multigrid method as smoothers. And we provide the basic knowledge on preconditioners for iterative methods. However, using multigrid as a preconditioner for the conjugate gradient is a relatively new approach motivated by engineering problems.

The multigrid method is a very effective iterative method for the mechanical analysis of heterogeneous material samples in [39]. However, the increase in the ratio of Young's moduli between matrix material and inclusion leads to a significantly worse condition number of the system, which slows the solution process. This could be also the result of the worse material representation on coarse grids. For a similar problem, Poisson's equation with large coefficient jumps or differences of grid spacing in coordinate transformation, the worse condition number will also lead to the slow solving process with the iterative methods mentioned above. As Poisson's equation is the key challenge in earthquake cycle simulation, an effective approach to solving linear systems with worse

condition numbers is worth exploring. It has been shown that the multigrid preconditioned conjugate gradient method has a superior convergence rate over the multigrid method as a solver [86]. This approach is less dependent on the considered problem.

The conditions of the multigrid preconditioners are examined in [86]. According to [91], the multigrid method will potentially provide a valid preconditioner if the smoother is symmetric. For a derivation of the preconditioned conjugate gradient method, we would introduce a matrix $\boldsymbol{L}$ which satisfies $\boldsymbol{M}^{-1} = \boldsymbol{L}^T\boldsymbol{L}$ as shown in [91] (Our notation $\boldsymbol{M}$ is equivalent to $\boldsymbol{H}$ in the paper). The Equation VII.8 improves the convergence if the condition number of the preconditioned matrix $\boldsymbol{M}^{-1}\boldsymbol{A}$ is lower than that of the original matrix $\boldsymbol{A}$, which can be determined from the analysis of eigenvalues as presented in [37]. If the preconditioning matrix is exactly $\boldsymbol{M}^{-1} = \boldsymbol{A}^{-1}$, the after one iteration step, the exact solution $\boldsymbol{u}$ is found. An ideal preconditioning matrix $\boldsymbol{M}^{-1}$ should be a reasonably close approximation of $\boldsymbol{A}^{-1}$. With respect to the initial search direction, the vector $\boldsymbol{p}^0 = \boldsymbol{M}^{-1}\boldsymbol{r}^0$ would correspond to the error $-\boldsymbol{e}^0$, if $\boldsymbol{M}^{-1} = \boldsymbol{A}^{-1}$. An adequate matrix $\boldsymbol{M}^{-1}$ leads to an improved initial search direction $\boldsymbol{p}^0$. Therefore, the preconditioned conjugate gradient method applies the following start conditions

$$\boldsymbol{r}^0 = \boldsymbol{f} - \boldsymbol{A}\boldsymbol{u}^0; \quad \tilde{\boldsymbol{r}}^0 = \boldsymbol{p}^0 = \boldsymbol{A}^{-1}\boldsymbol{r}^0 \tag{VII.36}$$

The following equations give a preconditioned conjugate gradient method adapted from [86] in the notation of the conjugate gradient method in subsection V-B.

$$\lambda_k = \frac{\tilde{\boldsymbol{r}}^{k^T}\boldsymbol{r}^k}{\boldsymbol{p}^{k^T}\boldsymbol{A}\boldsymbol{p}^k} \tag{VII.37}$$

$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \lambda_k\boldsymbol{p}^k \tag{VII.38}$$

$$\boldsymbol{r}^{k+1} = \boldsymbol{r}^k - \lambda_k\boldsymbol{A}\boldsymbol{p}^k \tag{VII.39}$$

$$\tilde{\boldsymbol{r}}^{k+1} = \boldsymbol{M}^{-1}\boldsymbol{r}^{k+1} \tag{VII.40}$$

$$\boldsymbol{p}^{k+1} = \tilde{\boldsymbol{r}}^{k+1} + \frac{\tilde{\boldsymbol{r}}^{k+1^T}\boldsymbol{r}^{k+1}}{\tilde{\boldsymbol{r}}^{k^T}\boldsymbol{r}^k}\boldsymbol{p}^k \tag{VII.41}$$

In each iteration step, preconditioning only takes place in Equation VII.40 and generates a new vector $\tilde{\boldsymbol{r}}^{k+1}$. The preconditioning matrix $\boldsymbol{M}^{-1}$ does not need to be explicitly built. The operation defined in Equation VII.40 can be replaced by a multigrid cycle that solves a linear system with $\boldsymbol{r}^{k+1}$ being the right hand side, and the solution is then assigned to $\tilde{\boldsymbol{r}}^{k+1}$. The preconditioned conjugate gradient method preserves a system of conjugate directions, while each increment is optimized for each improved search direction based on the multigrid method. Therefore, this optimization leads to considerably improved increments, if the stiffness of the coarse meshes is generally overestimated.

D. Machine-learned Preconditioner

The purpose of the preconditioner is to create a matrix M or an equivalent form that is approximate to the A^{-1} to accelerate the convergence of iterative methods. Machine learning has been applied to various fields of research to approximate any function or mapping given the input data and the output data. Recently, machine learning has been applied to solve PDEs which can be difficult to handle with numerical methods. This new approach of physics-informed machine learning (PIML) or specifically physics-informed neural networks (PINN) has gained a lot of attention over the past decade [48]. If a linear system can be solved efficiently with machine learning, then similar to the multigrid, this machine-learned solver can be used as a preconditioner for iterative methods. Machine learning can be used to generate sparsity patterns for traditional block-Jacobi preconditioners[33]. Machine learning can be also used to generate local approximate inverse preconditioners for studying geophysical fluid flows [33]. One nice feature of preconditioners obtained in this approach is that machine learning is usually built with frameworks like Pytorch or Tensorflow that support parallel processing on multiple different platforms. Thus these approaches are highly parallel by nature. However, integrating tools built in another high-level language can often be challenging for performance optimization if the main code is written in languages like C++/FORTRAN for high-performance computing. Preconditioners that are designed with parallel implementation in mind will be covered in subsection IX-A. More detailed parallel preconditioners will be covered in subsection IX-A.

VIII. Intial Guess for Iterative Methods

Preconditioners have been powerful tools to improve the performance of iterative methods by reducing the number of iterations. Another way to reduce the number of iterations is to use a good initial guess. It is not always easy to obtain good initial guesses for iterative methods, but in a time-dependent PDE simulation, using the solution computed at the previous time step as an initial guess for the solution at the current one is a natural choice. This requires no additional computation. An even better initial guess can be obtained by combining the solutions at several previous time steps instead of just one [76]. This method takes the linear combination of the previous solutions that minimize the norm of the residual when substituted into the linear system by orthogonal projections. This method requires growing storage costs as the simulation runs, however, this can be reduced by periodic restarts which discard the entire history space of stored solutions of the right-hand side vectors. Another way to avoid this issue is to use a classical technique for updating QR factorization[4]. This approach of obtaining initial guesses has been successfully applied to the study of incompressible flow, but similar ideas can be also applied to nuclear physics [18] and computational electromagnetics [22, 23].

Another method for using prior solutions to generate initial guesses is polynomial extrapolation. Solutions from previous time steps can be used to fit a polynomial curve, and the initial guess for the current time step can be obtained by evaluating the polynomial curve at the current time step. The most basic version of this method is to fit a simple (Lagrange) interpolant curve. Approximations based on Taylor expansions are also proposed [22]. Simply using a solution at the previous time step can be considered as a zeroth-order or constant interpolation of this technique. Extrapolation based on

higher-order polynomial interpolation has been also considered [18, 34, 69]. The results of polynomial extrapolation show the limitation of efficiency. A right-hand side projection is guaranteed to provide a reduction in the residual norm, while there is no guarantee for the extrapolation.

A recent work re-examined two different approaches in the context of PDE solvers running on modern HPC architectures that employ GPUs for acceleration. Because GPUs can be challenging to program effectively, the performances of two different approaches on GPUs are also explored [8]. In addition to the methods mentioned above, a *stabilized polynomial extrapolation* method based on solving a least-square problem instead of naive Lagrange interpolation has been proposed. The proposed extrapolation methods perform comparably to- and, in some cases, slightly better than-equivalent projection methods while requiring less storage, moving less data, and performing fewer operations that require global communications[8]. The scalability study of this method on multiple GPUs and compute nodes and HPC applications of other approaches for improving initial guesses are still a research gap. New findings in this field will certainly change how numerical algorithms are designed for modern HPC platforms.

2 All suggestions

IX. Parallel Implementations

A. Parallel Implementation of Iterative Methods

The iterative methods are ideal for their low memory requirements, and this becomes extremely important as the simulations in many fields of study have moved towards three-dimensional models. Another appealing part of iterative methods is that they are far easier to implement in parallel than sparse direct methods because they only require a small set of computational kernels. However, iterative methods are usually slower than direct methods, requiring suitable preconditioning techniques for accelerated convergence. The parallel aspect of preconditioners also becomes very important naturally.

This subsection gives a short overview of various parallel architectures as well as different types of operations in iterative methods that can be parallelized.

There are currently three leading architectures of parallel models around which modern parallel processors are designed. These are

- The shared-memory model
- Single-instruction-multiple-data (SIMD)
- The distributed memory message passing model

1) Shared memory computers: A shared memory computer has processors connected to a large global memory, and the address space is the same for all processors. Data stored in a large global memory is readily accessible to any processor. There are two possible implementations of shared memory machines:

- bus-based architectures
- switch-based architectures

These two architectures are illustrated in Figure 3 and Figure 4. So far, the bus-based model has been used more often. Buses are the backbone for communication between the different units of most computers, and usually have higher bandwidth for data I/O. The main reason why the bus-based model is more common is that the hardware involved in such implementation is simpler [1]. However, memory conflicts as well as the necessity to maintain data coherence can lead to worse performance. Moreover, shared memory computers can not take advantage of data locality in problems such as solving PDEs. Some machines can even have logically shared but physically distributed memory.

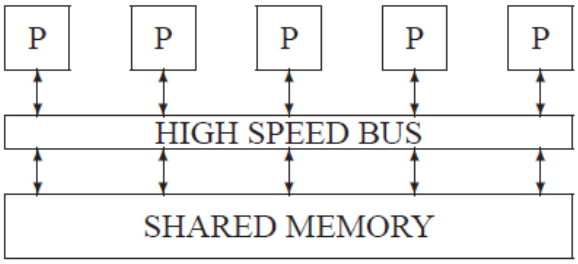

Fig. 3: A bus-based shared memory computer [75]

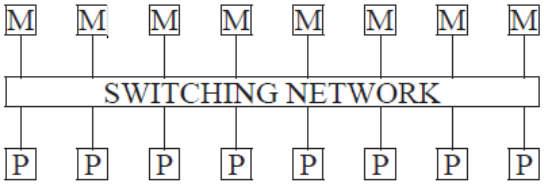

Fig. 4: A switch-based shared memory machine [75]

2) Distributed Memory Architectures: The *distributed memory* model can refer to distributed memory SIMD architecture or distributed memory with *memory passing* interface. A typical distributed memory system consists of a large number of identical processors and each processor has its own memory. These processors are interconnected in a regular topology. This can be shown with Figure 5. In these diagrams, each processor unit can be viewed as a complete processor with its one memory, CPU, I/O subsystems, control unit, etc. These processors are linked to a number of "neighboring" processors. In the "message passing" model, no global synchronizations are performed of the parallel tasks. Instead, computations are *data driven* because each processor performs a given task only when the operands it requires become available. The programmer needs to explicitly instruct data exchanges between different processors.

In the SIMD model, a different approach is used. A host processor stores the program and each slave processor holds different data. The host broadcasts instructions to each processor to execute them simultaneously. One advantage of this approach is that there is no need for large memories in each node to store the main program since the same instructions are broadcast to each processor.

Unlike the shared-memory model, distributed memory computers can easily exploit the data locality of data to reduce

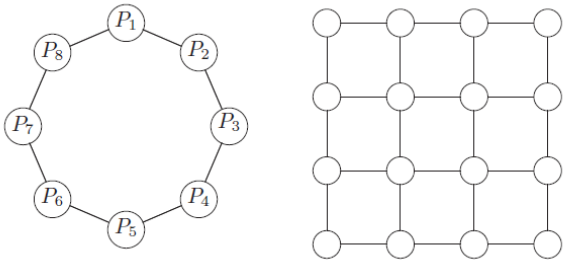

Fig. 5: An eight-processing ring (left) and 4 × 4 multiprocessor mesh (right) [75]

communication costs. Modern GPUs are designed with the SIMD model [68], and clusters with multiple CPUs are connected using a message passing interface (MPI) [9].

B. Key Operations in Parallel Implementation

1) Types of Operations: We use the preconditioned Conjugate Gradient to demonstrate the typical operations involved that can be parallelized. The PCG algorithm consists of the following types of operations:

- Preconditioner setup
- Matrix-vector multiplications
- Vector update
- Dot Product
- Preconditioning operations

Matrix-vector multiplications, vector updates, and dot products are common operations in so-called Basic Linear Algebra Subprograms (BLAS) that can be easily parallelized and have been well studied and implemented [26, 21, 30]. In terms of the computational costs, the vector update and dot product are much lower compared to the matrix-vector multiplication, which can still be carried out very quickly on the latest GPUs. The tricky part or the bottleneck for both memory and the runtime lies in the first step of preconditioner setup and the last step of preconditioning operations. We will discuss these two key operations in the next subsection. For now, let's focus on the Matrix-vector multiplication that has much higher computational costs than the vector update and the dot product.

2) Sparse Matrix-vector Products: The linear system coming from the discretization in the finite difference method is often sparse, which allows us to store them efficiently and use sparse matrix-vector products (SpMVs) for efficient computation. Different formats for storing sparse matrices can be found in [75]. Compressed Sparse Row (CSR) sparse matrix format is one of the earliest sparse formats developed. It is ideal for parallelization since the data from each row can be handled independently. The SpMV algorithm for CSR format as well as the demonstration of the storage scheme of the CSR format is shown in Figure 6

A summary of different sparse matrix formats in the following table as well as a detailed performance comparison of these different formats can be found in [79]. There are also recent work on developing new sparse matrix formats for

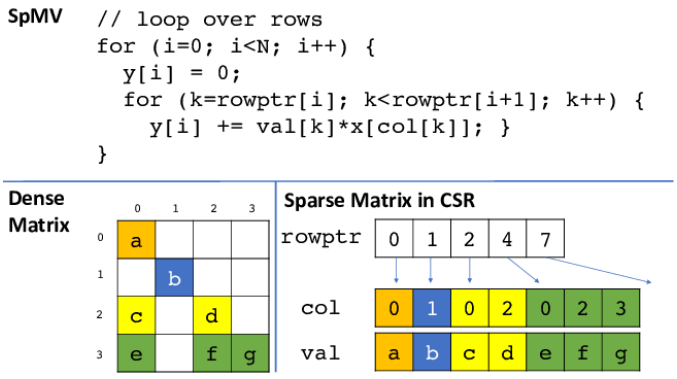

Fig. 6: The `val` array stores the nonzeros by packing each row in contiguous locations. The `rowptr` array points to the start of each row in the `val` array. The `col` array is parallel to `val` and maps each nonzero to the corresponding column.[65]

Short	Name	Short	Name
DNS	Dense	Ell	Ell-pack ItPack
BND	Linpack Banded	DIA	Diagonal
COO	Coordinate	BSR	Block Sparse Row
CSR	Compressed Sparse Row	SSK	Symmetric Skyline
CSC	Compressed Sparse column	BSR	Nonsymmetric Skyline
MSR	Modified CSR	JAD	Jagged Diagonal
LIL	Linked List		

TABLE II: A summary of different sparse matrix formats and their short names

optimal performance for different use cases [77, 27] or implementing SpMV algorithms on parallel architectures [12, 94, 56]. Using the right sparse matrix formats and implementing them on suitable architectures can reduce the time spent on these SpMV calculations significantly, which makes iterative methods run faster. Another way to accelerate these iterative methods is to use the preconditioners that we mentioned before. A parallel implementation of these preconditioners becomes more challenging because of the complex arithmetic operations compared to the SpMV or other BLAS operations. We will focus on the parallel preconditioning technique in the next subsection.

C. Parallel Preconditioning

1) Parallelism in Solving Linear Systems: Each preconditioned step from the previous subsection requires the solution of a linear system of equations of the form $Mz = y$. We consider traditional preconditioners such as ILU or SOR or SSOR, in which a solution with M is the result of solving triangular systems. Since these preconditioners are commonly used, it's important to explore their efficient parallel implementations for the iterative methods to be parallel. These preconditioners are mostly implemented on shared memory parameters. The distributed memory computers would use different strategies. These preconditioners require some sort of factorization, and the parallelism is done by sweeping through the lower triangular matrix or upper triangular matrix. Typical parallelism can be seen using a forward sweep.

It's typical for solving a lower triangular system that the solution is overwritten onto the right-hand side. So there is only one array u needed for both the solution and the right-hand side. The forward sweep for solving lower triangular systems with coefficients $A(i,j)$ and right-hand-side b is defined as follows:

Algorithm 1: Sparse Forward Elimination

```
for i = 2...n do
    for all j that A(i,j) is nonzero do
        u(i) ← u(i) − A(i,j) * u(j)
    end
end
```

Here $A(i,j)$ refers to the element in the sparse matrix. The different sparse matrix formats will have different implementations of locating this element, so the inner for loop will be implemented differently and the $A(i,j)$ will be replaced by different indexing code in different sparse matrix formats.

2) Parallel preconditioners: Several techniques can be for parallel implementations of the preconditioners. They can be summarized into three types of techniques. The simplest approach is to use a Jacobi or block Jacobi approach. In this case, a Jacobi preconditioner may be consist of a diagonal or a block-diagonal of A

To improve the performance, these preconditioners can be accelerated by polynomial iterations. For example, the second level of preconditioning is called *polynomial preconditioning*.

A different strategy is to enhance parallelism by using graph algorithms, such as graph-coloring techniques. The gist of this approach is that all unknowns associated with the same color can be determined simultaneously in the forward or backward sweeps.

The third strategy uses generalizations of "partitioning" techniques which can be also called "domain decomposition" approaches.

We will give a brief overview of these methods in this part.

Overlapping block-Jacobi preconditioning is a parallel preconditioner similar to the general block-Jacobi approach with overlapping blocks as shown in Figure 7. Enlarging a system of algebraic equations by including duplicate copies of several rows, leads to an efficient iterative scheme on a multiprocessor MIMD array [90].

Polynomial preconditioners are another family of parallel preconditioners. In polynomial preconditioners, the matrix $\boldsymbol{M}$ is defined by $\boldsymbol{M^{-1} = s(A)}$, where $\boldsymbol{s}$ is a polynomial, typically of low degree. Thus the original system can be preconditioned by

$$\boldsymbol{s(A)Au = s(A)f} \tag{IX.1}$$

Note that the $\boldsymbol{s(A)}$ and $\boldsymbol{A}$ commute, and as a result, the preconditioned matrix is the same for left or right preconditioning. In addition, the matrix $\boldsymbol{s(A)}$ or $\boldsymbol{As(A)}$ does not need to be formed explicitly in matrix form, which allows the use of matrix-free methods. This approach was initially motivated by the good performance of matrix-vector operations on vector computers. It has now become more popular on iterative

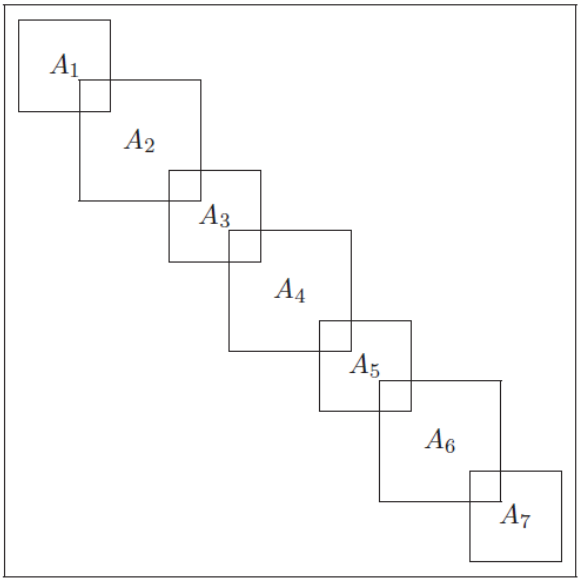

Fig. 7: The block-Jacobi matrix with overlapping blocks [75]

methods for GPU computing because of the similar SIMD architecture. There are several ways to construct polynomials in this method. The simplest polynomial s is the polynomial of the Neumann series expansion

$$I + N + N^2 + \cdots + N^s \tag{IX.2}$$

In which

$$N = I - \omega A \tag{IX.3}$$

ω is called the scaling parameter. The series above comes from expanding the inverse of ωA using the splitting

$$\omega A = I - (I - \omega A) \tag{IX.4}$$

This approach can be generalized using the splitting of the form

$$\omega A = D - (D - \omega A) \tag{IX.5}$$

where D can be a diagonal of A, or a block diagonal of A. Then

$$\begin{aligned}(\omega A)^{-1} &= [D(I - (I - \omega D^{-1}A))]^{-1} \\ &= [I - (I - \omega D^{-1}A)]^{-1}D^{-1}\end{aligned} \tag{IX.6}$$

Thus, setting

$$N = I - \omega D^{-1}A \tag{IX.7}$$

results in the approximate s-term expansion

$$(\omega A)^{-1} \approx M^{-1} = [I + N + \cdots + N^s]D^{-1} \tag{IX.8}$$

Since $D^{-1}A = \omega^{-1}[I - N]$, note that

$$\begin{aligned}M^{-1}A &= [I + N + \cdots + N^s]D^{-1}A \\ &= \frac{1}{\omega}[I + N + \cdots + N^2](I - N) \\ &= \frac{1}{\omega}(I - N^{s+1})\end{aligned} \tag{IX.9}$$

The matrix operation with the preconditioned matrix can be difficult numerically for large s. If the original matrix is SPD, then even though $M^{-1}A$ is not symmetric, it is self-adjoint with respect to the $D-inner$ product.

The polynomial s can be selected to be optimal in some sense, and this leads to the Chebyshev polynomials. The criterion used to make the preconditioned matrix $s(A)A$ as close as possible to the identity matrix in some sense. For example, the spectrum of the preconditioned matrix can be made as close as possible to that of the identity. We denote the spectrum of A by $\sigma(A)$, and by $\mathbb{P}_k$ the space of polynomials of degree not exceeding k. We need to solve the following problem

$$\text{Find} s \in \mathbb{P}_k \text{ which minimizes:} \tag{IX.10}$$

$$\max_{\lambda \in \sigma(A)} |1 - \lambda s(\lambda)| \tag{IX.11}$$

This problem involves all eigenvalues of A, and is hard to solve than the original problem. It can be replaced by the following problem

$$\text{Find} s \in \mathbb{P}_k \text{ which minimizes:} \tag{IX.12}$$

$$\max_{\lambda \in E} |1 - \lambda s(\lambda)| \tag{IX.13}$$

which is obtained by replacing the set $\sigma(A)$ by some continuous set E that encloses it. Thus, a rough idea of the spectrum of matrix A is needed. If A is SPD, then E can be taken as an interval $[\alpha, \beta]$ containing the eigenvalues of A. A variation of the approximation theorem says for any real scalar γ such that $\gamma \leq \alpha$ the minimum

$$\min_{p \in \mathbb{P}_k, p(\gamma)=1} \max_{t \in [\alpha,\beta]} |p(t)| \tag{IX.14}$$

is reached for the shifted and scaled Chebyshev polynomial of the first kind

$$\hat{C}_k(t) = \frac{C_k(1 + 2\frac{\alpha - t}{\beta - \alpha})}{C_k(1 + 2\frac{\alpha - \gamma}{\beta - \alpha})} \tag{IX.15}$$

When $\gamma = 0$, this gives the polynomial

$$T_k(t) = \frac{1}{\sigma_k} C_k(\frac{\beta + \alpha - 2t}{\beta - \alpha}) \text{ with } \sigma_k = C_k(\frac{\beta + \alpha}{\beta - \alpha}) \tag{IX.16}$$

The Chebyshev iteration can be found in [75]. One nice feature of the Chebyshev iteration is that it does not require inner products, and this is very attractive for parallel implementation as it does not require reductions.

Other polynomials include least-squares polynomials. A comparison of Chebyshev polynomials and least-square polynomials can be found in [6]. So far, Chebyshev polynomials have been the most popular for parallel implementation, especially in the matrix-free setting where the assembly of the matrix can be very expensive.

Multicolor preconditioners are similar to ILU preconditioners in the sense that the construction and factorization of the matrices are required. Methods like these can be done in parallel, but they are not suitable for GPUs.

D. CUDA architecture and CUDA

Given the increasing importance and popularity of GPUs in modern supercomputers, this subsection is dedicated to GPU architecture. As NVIDIA GPUs are mostly used in the industry for scientific computing and machine learning, the GPU programming model will be focused on CUDA (Compute Unified Device Architecture) toolkit.

A GPU is built as a scalable array of multithreaded *Streaming Multiprocessors* (SMs), each of which consists of multiple *Scalar Processor* (SP) cores. To manage hundreds or thousands of threads, the multiprocessors employ a *Single Instruction Multiple Threads* (SIMT) model with each thread mapped into one SP core and executing independently with its own instruction address and register state [92].

The NVIDIA GPU platform has various memory architectures. The types of memory can be classified as follows:

- off-chip global memory
- off-chip local memory
- on-chip shared memory
- read-only cached off-chip constant memory and texture memory
- registers

The effective bandwidth of each type of memory depends significantly on the access pattern. Global memory is relatively large but has a much higher latency. Using the right access pattern such as memory coalescing and avoiding bank conflicts will help achieve good memory bandwidth.

Threads are organized in *warps*. A warp is defined as a group of 32 threads of consecutive thread IDs. More detailed information on optimizing memory access patterns can be found in [92].

GPUs were initially designed for graphics-related calculations such as image rendering. General-purpose GPU programming on NVIDIA GPUs is supported by the NVIDIA CUDA toolkit. CUDA programs use similar syntax to C++. The main code on the host (CPU) would invoke a *kernel grid* that runs on the device (GPU). The same parallel kernel is executed by many threads. These threads are organized into thread blocks. Blocks and threads are the logical division of the GPU and are mapped to the actual SMs. Thread blocks are split into warps scheduled by SIMT units. All threads in the same block share the same shared memory and can be synchronized by a barrier. Threads in a warp execute one common instruction at a time. This is referred as warp-level synchronization [92]. It's most efficient when 32 threads of a warp follow the same execution path. Branch divergence in which threads within the same warp are executing different instructions often causes worse performance/

CUDA is only a lower-level tool for direct kernel programming. Libraries built on top of CUDA allow users to directly use code and kernels written for different tasks without manually programming and optimizing kernels themselves. Existing common CUDA libraries that supports GPU SpMV operation include CUDPP (CUDA Data Parallel Primitives)[40], NVIDIA Cusp library [25], and the IBM SpMV library [10]. In these packages, different formats of sparse matrices are studied for producing high-performance SpMV kernels on GPUs.

These include the compressed sparse row (CSR) format, the coordinate format (COO), the diagonal (DIA) format, the ELLPACK (ELL) format., and a hybrid (ELL/COO) format. There are other recent sparse matrix formats specifically designed for GPU computing, but we will not go into detail to cover each of them.

For dense linear algebra computations, the MAGMA (Matrix Algebra for GPU and Multicore Architectures) project hybrid multicore-multi-GPU system aims to develop a dense linear algebra similar to LAPACK[3]. Since our numerical methods for PDEs would generate a sparse linear system, we did not explore this library in this paper.

E. A Review of Different Parallel Preconditioners on GPUs

As we are interested in solving PDEs on GPUs due to their high memory bandwidth and FLOPS. We want to focus on the preconditioners that are not only parallelizable but also suitable for the GPU architecture. A detailed overview of GPU-accelerated preconditioners can be found in [57]. We summarize the parallel preconditioners mentioned in this paper here with an emphasis on their compatibility with GPUs.

1) ILU/IC Preconditioners: The Incomplete LU(ILU) factorization process for a sparse matrix $\boldsymbol{A}$ computes a sparse lower triangular matrix $\boldsymbol{L}$ and a sparse upper triangular matrix $\boldsymbol{U}$ such that the residual matrix $\boldsymbol{R} = \boldsymbol{LU} - \boldsymbol{A}$ satisfy certain conditions [75]. Since the preconditioning matrix $\boldsymbol{M} = \boldsymbol{LU}$ and $\boldsymbol{LU} \approx \boldsymbol{A}$, the preconditioning operation is a back substitution followed by forward substitution. i.e. $\boldsymbol{u} = \boldsymbol{M}^{-1}\boldsymbol{v} = \boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{v}$. To retain the sparsity of the original matrix $\boldsymbol{A}$, we sometimes forbid certain fill-ins for the ILU process. If no fill-in is allowed in the ILU process, we obtain the so-called ILU(0) preconditioner, which has the same sparsity pattern as $\boldsymbol{A}$. However, this is achieved by sacrificing accuracy. We can improve the accuracy by allowing more fill-ins in ILU(k) factorization or ILUT factorization that uses an alternative method to drop fill-ins [46]. When $\boldsymbol{A}$ is SPD, the incomplete Cholesky (IC) factorization. However, the IC factorization for an SPD matrix does not always exist. The modified incomplete Cholesky (MIC) is proposed for any SPD matrix [70]. The performance of a triangular solver on GPUs deteriorates severely when the number of scheduling levels increases. Triangular systems obtained from ILU or IC factorization with many fill-ins usually have a large number of levels. As a result, applying ILU/IC preconditioners may not be a good choice.

2) Block Jacobi Preconditioners: The simplest parallel preconditioner might be the block Jacobi technique, which breaks up the system into independent local sub-systems. These subsystems correspond to the diagonal blocks in the reordered matrix. These subsystems correspond to the diagonal blocks in the reordered matrix. The global solution is obtained by summing up solutions of all local systems, and the local systems can be solved approximately by the ILU factorization. The setup and application phases of the preconditioner can be performed locally without data dependency. Therefore, we can use one CUDA thread block for each local block without requiring global synchronization or communications. Threads within each thread block can solve the local system in parallel and save the local results to the global output vector.

Although a block Jacobi preconditioner with a large number of blocks can provide high parallelism, a well-known drawback is that it usually requires a much larger number of iterations to converge. However, this drawback can be outweighed by the increased performance on new GPU architectures with more computing power which makes highly parallel algorithms more appealing than algorithms with fewer computations but bad or parallel implementations.

3) Multicolor SSOR/ILU(0) preconditioners: Multicolor SSOR/ILU(0) preconditioners improve the parallelism in the SSOR/ILU(0) preconditioners by multicolor reordering. The parallelism is of order n/p where p is the number of colors.

4) Polynomial Preconditioners: As we mentioned in the previous section, the polynomial preconditioners are nice because they do not require the construction of matrix $\boldsymbol{A}$, and the polynomial matrix $\boldsymbol{s}(A)$ can be obtained via a sequence of SpMV operations. This is ideal for computations on GPUs and for finite difference methods where the SpMV can be replaced by matrix-free stencil computation. This method can produce a good approximation of $\boldsymbol{A}^{-1}$ assuming that a tight bound of smallest eigenvalues λ_n and the largest eigenvalue λ can be found. As mentioned in [97], the upper bound must be larger than λ, but it should not be too large as this would otherwise result in slower convergence. The simplest technique is to use Gershgorin's theorem mentioned in subsection VII-B, but bounds obtained in this way are usually big overestimates of λ. A much better estimation can be obtained inexpensively by using a small number of steps of the Lanczos method [55]. A safeguard term is added to the "quasi"-guarantee that the upper bound is larger than $\rho(A)$. Typically, a small number of steps in Lanczos iterations are enough to provide a good estimate of extreme eigenvalues. And the Lanczos algorithm can be efficiently implemented in GPU as well.

X. Summary

In this work, we reviewed different iterative methods for solving earthquake cycle simulations. We start by describing the background of the earthquake cycle simulations and the mathematical problem that we are trying to solve, which is the classical Poisson's equation. We use the SBP-SAT finite difference method for discretization. The linear system generated from the SBP-SAT method is sparse and SPD, making them ideal for iterative methods. We examined different traditional static iterative methods as well as Krylov subspace methods with CG as an example. Multigrid is a powerful framework for solving elliptic PDEs. It can be used as a solver itself. The study of the multigrid focus on its connection with the SBP-SAT method, which introduces SBP-preserving interpolation techniques for the multigrid method. Since iterative methods can be also used as preconditioners for the linear system. We dedicated a section to cover the theory and convergence analysis for different iterative methods. In this section, we mainly use classical iterative methods to showcase the different applications of the general convergence theory. The general convergence theory can be also applied

to other iterative methods such as Krylov subspace methods and Multigrid methods. We used a multigrid preconditioned conjugate gradient as an example. The convergence analysis of this approach can be found in the paper that we cited as well as much recent research on the convergence of multigrid. Since we need to solve a large linear system, we did a review of HPC techniques with a focus on GPU computing and connected these different iterative methods and preconditioners with GPU computing in the last section. We would like to use this work to summarize the common techniques related to our research to give us a better direction for solving a large-scale earthquake cycle simulation that hasn't been well studied but will certainly have huge societal impacts. Research from this project can be also applied to other related fields where solving a large linear system on GPUs with different approaches is the bottleneck for simulation.

References

[1] ADELI, HOJJAT and VISHNUBHOTLA, PRASAD. "Parallel processing". *Computer-Aided Civil and Infrastructure Engineering* 2.3 (1987), pp. 257–269.

[2] Adler, James H et al. "Monolithic multigrid methods for two-dimensional resistive magnetohydrodynamics". *SIAM Journal on Scientific Computing* 38.1 (2016), B1–B24.

[3] Agullo, Emmanuel et al. "Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects". *Journal of Physics: Conference Series*. Vol. 180. 1. IOP Publishing. 2009, p. 012037.

[4] Ali, Mouhamad Al Sayed and Sadkane, Miloud. "Improved predictor schemes for large systems of linear ODEs". *Electronic Transactions on Numerical Analysis* 39 (2012), pp. 253–270.

[5] Ariyoshi, Keisuke et al. "Influence of interaction between small asperities on various types of slow earthquakes in a 3-D simulation for a subduction plate boundary". English. *Gondwana Research* 16.3-4 (Dec. 2009). Funding Information: Discussion with Dr. Bunichiro Shibazaki was useful to sharpen the target of our simulation. Dr. Kazushige Obara kindly taught us the details of observed low-frequency earthquake behaviors. We thank two reviewers for constructive comments and Dr. M. Santosh and Dr. S. Maruyama for expeditious editorial handling. This study used the Earth Simulator for our simulations, and the supercomputing resources at Cyberscience Center of Tohoku University for running experimental simulation to improve program codes. This work is supported by DONET program of the Ministry of Education, Culture, Sports, Science and Technology. GMT software (Wessel and Smith, 1998) was used to draw some figures., pp. 534–544. ISSN: 1342-937X. DOI: 10.1016/j.gr.2009.03.006.

[6] Ashby, Steven F, Manteuffel, Thomas A, and Otto, James S. "A comparison of adaptive Chebyshev and least squares polynomial preconditioning for Hermitian positive definite linear systems". *SIAM Journal on Scientific and Statistical Computing* 13.1 (1992), pp. 1–29.

[7] Asmar, Nakhlé H. *Partial differential equations with Fourier series and boundary value problems*. Courier Dover Publications, 2016.

[8] Austin, Anthony P, Chalmers, Noel, and Warburton, Tim. "Initial Guesses for Sequences of Linear Systems in a GPU-Accelerated Incompressible Flow Solver". *SIAM Journal on Scientific Computing* 43.4 (2021), pp. C259–C289.

[9] Barker, Brandon. "Message passing interface (mpi)". *Workshop: High Performance Computing on Stampede*. Vol. 262. Cornell University Publisher Houston, TX, USA. 2015.

[10] Baskaran, Muthu Manikandan and Bordawekar, Rajesh. "Optimizing sparse matrix-vector multiplication on GPUs". *IBM research report RC24704* W0812–047 (2009).

[11] Bathe, Klaus-Jürgen. *Finite element procedures*. Klaus-Jurgen Bathe, 2006.

[12] Bell, N. and Garland, M. "Implementing sparse matrix-vector multiplication on throughput-oriented processors". *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. 2009, pp. 1–11.

[13] Bell, Nathan, Olson, Luke N, and Schroder, Jacob. "Pyamg: algebraic multigrid solvers in python". *Journal of Open Source Software* 7.72 (2022), p. 4142.

[14] Bielak, Jacobo, Ghattas, Omar, and Kim, EJ. "Parallel octree-based finite element method for large-scale earthquake ground motion simulation". *Computer Modeling in Engineering and Sciences* 10.2 (2005), p. 99.

[15] Brandt, Achi. "Algebraic multigrid theory: The symmetric case". *Applied mathematics and computation* 19.1-4 (1986), pp. 23–56.

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

[17] Briggs, William L., Henson, Van Emden, and McCormick, Steve F. *A Multigrid Tutorial (2nd Ed.)* USA: Society for Industrial and Applied Mathematics, 2000. ISBN: 0898714621.

[18] Brower, Richard C et al. "Chronological inversion method for the Dirac matrix in hybrid Monte Carlo". *Nuclear Physics B* 484.1-2 (1997), pp. 353–374.

[19] Carpenter, M. H., Gottlieb, D., and Abarbanel, S. "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 (1994), pp. 220–236. DOI: 10.1006/jcph.1994.1057.

[20] Carpenter, M. H., Nordström, J., and Gottlieb, D. "A stable and conservative interface treatment of arbitrary spatial accuracy". *Journal of Computational Physics* 148.2 (1999), pp. 341–365. DOI: 10.1006/jcph.1998.6114.

[21] Chtchelkanova, Almadena et al. "Parallel implementation of BLAS: General techniques for level 3 BLAS". *Concurrency: Practice and Experience* 9.9 (1997), pp. 837–857.

[22] Clemens, Markus, Wilke, Marcus, and Weiland, Thomas. "Extrapolation strategies in numerical schemes for transient magnetic field simulations". *IEEE transactions on magnetics* 39.3 (2003), pp. 1171–1174.

[23] Clemens, Markus et al. "Subspace projection extrapolation scheme for transient field simulations". *IEEE Transactions on Magnetics* 40.2 (2004), pp. 934–937.

[24] Currenti, Gilda, Del Negro, Ciro, and Ganci, Gaetana. "Modelling of ground deformation and gravity fields using finite element method: an application to Etna volcano". *Geophysical Journal International* 169.2 (2007), pp. 775–786.

[25] Dalton, Steven et al. "Cusp: Generic parallel algorithms for sparse matrix and graph computations". *Version 0.5. 0* (2014).

[26] Dongarra, J. J. et al. "A Set of Level 3 Basic Linear Algebra Subprograms". *ACM Trans. Math. Softw.* 16.1 (Mar. 1990), pp. 1–17. ISSN: 0098-3500. DOI: 10.1145/77626.79170. URL: https://doi.org/10.1145/77626.79170.

[27] Dongarraxz, Jack et al. "A sparse matrix library in C++ for high performance architectures". *Second object oriented numerics conference*. 1994, pp. 214–218.

[28] Erickson, B. A. and Dunham, E. M. "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 (2014), pp. 3290–3316.

[29] Fedorenko, Radii Petrovich. "Iterative methods for elliptic difference equations". *Russian Mathematical Surveys* 28.2 (1973), pp. 129–195.

[30] Freeman, T Len and Phillips, Chris. *Parallel numerical algorithms*. Prentice-Hall, Inc., 1992.

[31] Gazdag, Jenö. "Wave equation migration with the phase-shift method". *GEOPHYSICS* 43.7 (1978), pp. 1342–1351. DOI: 10.1190/1.1440899. eprint: https://doi.org/10.1190/1.1440899. URL: https://doi.org/10.1190/1.1440899.

[32] Gee, Michael W et al. *ML 5.0 smoothed aggregation user's guide*. Tech. rep. Technical Report SAND2006-2649, Sandia National Laboratories, 2006.

[33] Götz, Markus and Anzt, Hartwig. "Machine learning-aided numerical linear algebra: Convolutional neural networks for the efficient preconditioner generation". *2018 IEEE/ACM 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (scalA)*. IEEE. 2018, pp. 49–56.

[34] Grinberg, Leopold and Karniadakis, George Em. "Extrapolation-based acceleration of iterative solvers: Application to simulation of 3D flows". *Communications in Computational Physics* 9.3 (2011), pp. 607–626.

[35] Grossmann, Ch. "Hackbusch, W., Iterative Lösung großer schwach besetzter Gleichungssysteme. Stuttgart, B. G. Teubner 1991. 382 S., DM 42,- ISBN 3-519-02372-5 (Leitfaden der angewandten Mathematik und Mechanik 69, Teubern-Studienbücher: Mathematik)". *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik* 74.2 (1994), pp. 96–96. DOI: https://doi.org/10.1002/zamm.19940740205. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/zamm.19940740205. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/zamm.19940740205.

[36] Gustafsson, Bertil, Kreiss, Heinz-Otto, and Oliger, Joseph. *Time dependent problems and difference methods*. Vol. 24. John Wiley & Sons, 1995.

[37] Hackbusch, Wolfgang. *Iterative Lösung großer schwachbesetzter Gleichungssysteme*. Vol. 69. Springer-Verlag, 2013.

[38] Hackbusch, Wolfgang. *Multi-grid methods and applications*. Vol. 4. Springer Science & Business Media, 2013.

[39] Häfner, Stefan et al. "Mesoscale modeling of concrete: Geometry and numerics". *Computers & structures* 84.7 (2006), pp. 450–461.

[40] Harris, Mark et al. *CUDPP: CUDA data parallel primitives library*. 2007.

[41] Hestenes, Magnus Rudolph, Stiefel, Eduard, et al. *Methods of conjugate gradients for solving linear systems*. Vol. 49. 1. NBS Washington, DC, 1952.

[42] Hesthaven, Jan S and Warburton, Tim. *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*. Springer Science & Business Media, 2007.

[43] Hillers, G. et al. "Statistical properties of seismicity of fault zones at different evolutionary stages". *Geophysical Journal International* 169.2 (May 2007), pp. 515–533. ISSN: 0956-540X. DOI: 10.1111/j.1365-246X.2006.03275.x. eprint: https://academic.oup.com/gji/article-pdf/169/2/515/5905493/169-2-515.pdf. URL: https://doi.org/10.1111/j.1365-246X.2006.03275.x.

[44] Hirose, Hitoshi and Obara, Kazushige. "Repeating short- and long-term slow slip events with deep tremor activity around the Bungo channel region, southwest Japan". *Earth, Planets and Space* 57.10 (Oct. 2005), pp. 961–972. ISSN: 1880-5981. DOI: 10.1186/BF03351875. URL: https://doi.org/10.1186/BF03351875.

[45] Hori, Muneo et al. "Application of HPC to Earthquake Hazard and Disaster Estimation". *Sustained Simulation Performance 2014*. Ed. by Michael M. Resch et al. Cham: Springer International Publishing, 2015, pp. 203–220. ISBN: 978-3-319-10626-7.

[46] Hysom, David and Pothen, Alex. "Efficient parallel computation of ILU (k) preconditioners". *Proceedings of the 1999 ACM/IEEE conference on Supercomputing*. 1999, 29–es.

[47] Jagota, Vishal, Sethi, Aman Preet Singh, and Kumar, Khushmeet. "Finite element method: an overview". *Walailak Journal of Science and Technology (WJST)* 10.1 (2013), pp. 1–8.

[48] Karniadakis, George Em et al. "Physics-informed machine learning". *Nature Reviews Physics* 3.6 (2021), pp. 422–440.

[49] Kato, Naoyuki. "Numerical simulation of recurrence of asperity rupture in the Sanriku region, northeastern Japan". *Journal of Geophysical Research: Solid Earth* 113.B6 (2008). DOI: https : / / doi . org / 10 . 1029 / 2007JB005515. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2007JB005515. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2007JB005515.

[50] Kawahara, Matsuto, Takeuchi, Norio, and Yoshida, Takaharu. "Two step explicit finite element method for tsunami wave propagation analysis". *International Journal for Numerical Methods in Engineering* 12.2 (1978), pp. 331–351.

[51] Kozdon, J. E., Erickson, B. A., and Wilcox, L. C. "Hybridized Summation-By-Parts Finite Difference Methods". *in press in the Journal of Scientific Computing* (2021).

[52] Kreiss, H.-O. and Scherer, G. "Finite element and finite difference methods for hyperbolic partial differential equations". *Mathematical aspects of finite elements in partial differential equations; Proceedings of the Symposium*. Madison, WI, 1974, pp. 195–212.

[53] Kreiss, H.-O. and Scherer, G. *On the existence of energy estimates for difference approximations for hyperbolic systems*. Tech. rep. Department of Scientific Computing, Uppsala University, 1977.

[54] Krylov, Aleksey Nikolaevich. "On the numerical solution of the equation by which in technical questions frequencies of small oscillations of material systems are determined". *Izvestija AN SSSR (News of Academy of Sciences of the USSR), Otdel. mat. i estest. nauk* 7.4 (1931), pp. 491–539.

[55] Lanczos, Cornelius. "An iteration method for the solution of the eigenvalue problem of linear differential and integral operators" (1950).

[56] Li, Kenli, Yang, Wangdong, and Li, Keqin. "Performance analysis and optimization for SpMV on GPU using probabilistic modeling". *IEEE Transactions on Parallel and Distributed Systems* 26.1 (2014), pp. 196–205.

[57] Li, Ruipeng and Saad, Yousef. "GPU-accelerated preconditioned iterative linear solvers". *The Journal of Supercomputing* 63.2 (2013), pp. 443–466.

[58] Lubarda, M. and Lubarda, V. *Intermediate Solid Mechanics*. DOI: 10.1017/9781108589000.011.

[59] Mandel, Jan. "Algebraic study of multigrid methods for symmetric, definite problems". *Applied mathematics and computation* 25.1 (1988), pp. 39–56.

[60] Mattsson, K. "Summation by parts operators for finite difference approximations of second-derivatives with variable coefficients". *Journal of Scientific Computing* 51.3 (2012), pp. 650–682. DOI: 10.1007/s10915-011-9525-z.

[61] Mattsson, K., Ham, F., and Iaccarino, G. "Stable Boundary Treatment for the Wave Equation on Second-Order Form". *Journal of Scientific Computing* 41.3 (2009), pp. 366–383.

[62] Mattsson, K. and Nordström, J. "Summation by parts operators for finite difference approximations of second derivatives". *Journal of Computational Physics* 199.2 (2004), pp. 503–540.

[63] Mattsson, Ken. "Boundary Procedures for Summation-by-Parts Operators". *Journal of Scientific Computing* 18.1 (Feb. 2003), pp. 133–153.

[64] McCormick, SF and Ruge, JW. "Multigrid methods for variational problems". *SIAM Journal on Numerical Analysis* 19.5 (1982), pp. 924–929.

[65] Mohammadi, Mahdi Soltan et al. "Sparse Matrix Code Dependence Analysis Simplification at Compile Time". *arXiv preprint arXiv:1807.10852* (2018).

[66] Nissen, A., Kreiss, G., and Gerritsen, M. "High Order Stable Finite Difference Methods for the Schrödinger Equation". *Journal of Scientific Computing* 55.1 (2013), pp. 173–199. DOI: 10.1007/s10915-012-9628-1.

[67] Ohtani, M. et al. "Fast Computation of Quasi-Dynamic Earthquake Cycle Simulation with Hierarchical Matrices". *Procedia Computer Science* 4 (2011). Proceedings of the International Conference on Computational Science, ICCS 2011, pp. 1456–1465. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2011.04.158. URL: https://www.sciencedirect.com/science/article/pii/S187705091100216X.

[68] Owens, John D et al. "GPU computing". *Proceedings of the IEEE* 96.5 (2008), pp. 879–899.

[69] Pitton, Giuseppe and Heltai, Luca. "Accelerating the iterative solution of convection–diffusion problems using singular value decomposition". *Numerical Linear Algebra with Applications* 26.1 (2019), e2211.

[70] Robert, Yves. "Regular incomplete factorizations of real positive definite matrices". *Linear Algebra and its Applications* 48 (1982), pp. 105–117.

[71] Ruge, John W and Stüben, Klaus. "Algebraic multigrid". *Multigrid methods*. SIAM, 1987, pp. 73–130.

[72] Ruggiu, Andrea A, Weinerfelt, Per, and Nordström, Jan. "A new multigrid formulation for high order finite difference methods on summation-by-parts form". *Journal of Computational Physics* 359 (2018), pp. 216–238.

[73] Ruggiu, Andrea A., Weinerfelt, Per, and Nordström, Jan. "A new multigrid formulation for high order finite difference methods on summation-by-parts form". *Journal of Computational Physics* 359 (2018), pp. 216–238. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.01.011. URL: https://www.sciencedirect.com/science/article/pii/S0021999118300214.

[74] Saad, Yousef. *Iterative Methods for Sparse Linear Systems*. Second. Society for Industrial and Applied Mathematics, 2003. DOI: 10.1137/1.9780898718003. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9780898718003. URL: https://epubs.siam.org/doi/abs/10.1137/1.9780898718003.

[75] Saad, Yousef. *Iterative methods for sparse linear systems*. SIAM, 2003.

[76] Simoncini, Valeria and Gallopoulos, Efstratios. "An iterative method for nonsymmetric systems with multiple

right-hand sides". *SIAM Journal on Scientific Computing* 16.4 (1995), pp. 917–933.

[77] Smailbegovic, FS, Gaydadjiev, Georgi N, and Vassiliadis, Stamatis. "Sparse matrix storage format". *Proceedings of the 16th Annual Workshop on Circuits, Systems and Signal Processing*. 2005, pp. 445–448.

[78] Sobhaninejad, G., Hori, M., and Kabeyasawa, T. "Enhancing Integrated Earthquake Simulation with High Performance Computing". *Adv. Eng. Softw.* 42.5 (May 2011), pp. 286–292. ISSN: 0965-9978. DOI: 10.1016/j.advengsoft.2010.10.009. URL: https://doi.org/10.1016/j.advengsoft.2010.10.009.

[79] Stanimirovic, Ivan P and Tasic, Milan B. "Performance comparison of storage formats for sparse matrices". *Ser. Mathematics and Informatics* 24.1 (2009), pp. 39–51.

[80] Stauft, D. L. "COMPUTER APPLICATIONS IN AN OIL-EXPLORATION COMPANY1". *Bulletin of Canadian Petroleum Geology* 16.1 (Mar. 1968), pp. 64–86. ISSN: 0007-4802. DOI: 10.35767/gscpgbull.16.1.064. eprint: https://pubs.geoscienceworld.org/bcpg/article-pdf/16/1/64/4957255/cspg_1968_0016_0001_0064.pdf. URL: https://doi.org/10.35767/gscpgbull.16.1.064.

[81] Stolk, Christiaan C, Ahmed, Mostak, and Bhowmik, Samir Kumar. "A multigrid method for the Helmholtz equation with optimized coarse grid corrections". *SIAM Journal on Scientific Computing* 36.6 (2014), A2819–A2841.

[82] Strand, B. "Summation by parts for finite difference approximations for d/dx". *Journal of Computational Physics* 110.1 (1994), pp. 47–67.

[83] Sundar, Hari, Stadler, Georg, and Biros, George. "Comparison of multigrid algorithms for high-order continuous finite element discretizations". *Numerical Linear Algebra with Applications* 22.4 (2015), pp. 664–680.

[84] Suppasri, Anawat et al. "Improvement of Tsunami Countermeasures Based on Lessons from The 2011 Great East Japan Earthquake and Tsunami — Situation After Five Years". *Coastal Engineering Journal* 58.4 (2016), pp. 1640011-1-1640011–30. DOI: 10.1142/S0578563416400118. eprint: https://doi.org/10.1142/S0578563416400118. URL: https://doi.org/10.1142/S0578563416400118.

[85] Svärd, M. and Nordström, J. "Review of summation-by-parts schemes for initial-boundary-value problems". *Journal of Computational Physics* 268 (2014), pp. 17–38.

[86] Tatebe, Osamu. "The multigrid preconditioned conjugate gradient method". *NASA conference publication*. NASA. 1993, pp. 621–621.

[87] Trottenberg, Ulrich, Oosterlee, Cornelius W, and Schuller, Anton. *Multigrid*. Elsevier, 2000.

[88] Vaněk, Petr, Mandel, Jan, and Brezina, Marian. "Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems". *Computing* 56.3 (1996), pp. 179–196.

[89] Virta, K. and Mattsson, K. "Acoustic wave propagation in complicated geometries and heterogeneous media". *Journal of Scientific Computing* 61.1 (2014), pp. 90–118. DOI: 10.1007/s10915-014-9817-1.

[90] Wait, R. and Brown, N.G. "Overlapping block methods for solving tridiagonal systems on transputer arrays". *Parallel Computing* 8.1 (1988). Proceedings of the International Conference on Vector and Parallel Processors in Computational Science III, pp. 325–333. ISSN: 0167-8191. DOI: https://doi.org/10.1016/0167-8191(88)90136-6. URL: https://www.sciencedirect.com/science/article/pii/0167819188901366.

[91] Wesseling, Pieter. "An Introduction to Multigrid Methods. RT Edwards". *Inc.,* 2 (2004), p. 2.

[92] Wilt, Nicholas. *The cuda handbook: A comprehensive guide to gpu programming*. Pearson Education, 2013.

[93] Xu, Jinchao and Zikatanov, Ludmil. "Algebraic multigrid methods". *Acta Numerica* 26 (2017), pp. 591–721.

[94] Yan, Shengen et al. "yaSpMV: Yet another SpMV framework on GPUs". *Acm Sigplan Notices* 49.8 (2014), pp. 107–118.

[95] Yang, Ulrike Meier et al. "BoomerAMG: A parallel algebraic multigrid solver and preconditioner". *Applied Numerical Mathematics* 41.1 (2002), pp. 155–177.

[96] Young, David M. *Iterative solution of large linear systems*. Elsevier, 2014.

[97] Zhou, Yunkai et al. "Parallel self-consistent-field calculations via Chebyshev-filtered subspace acceleration". *Physical Review E* 74.6 (2006), p. 066704.