HIGH PERFORMANCE COMPUTING METHODS FOR EARTHQUAKE CYCLE SIMULATIONS

by

ALEXANDRE CHEN

A DISSERTATION

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

June 2024

DISSERTATION APPROVAL PAGE

Student: Alexandre Chen

Title: High Performance Computing Methods for Earthquake Cycle Simulations

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

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

and

Krista M. Chronister	Interim Vice Provost for Graduate Studies

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

Degree awarded June 2024

DISSERTATION ABSTRACT

Alexandre Chen

Doctor of Philosophy

Department of Computer Science

June 2024

Title: High Performance Computing Methods for Earthquake Cycle Simulations

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

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

CURRICULUM VITAE

NAME OF AUTHOR: Alexandre Chen

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

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

DEGREES AWARDED:

Bachelor of Science, Physics, 2018, Fudan University

AREAS OF SPECIAL INTEREST:

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

PROFESSIONAL EXPERIENCE:

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

GRANTS, AWARDS AND HONORS:

Graduate Teaching/Research Fellowship, University of Oregon

PUBLICATIONS:

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

ACKNOWLEDGEMENTS

Here is an acknowledgment

To so-and-so...

TABLE OF CONTENTS

Chapter Page

Chapter Page

LIST OF FIGURES

Figure Page

LIST OF TABLES

Table Page

CHAPTER I

INTRODUCTION

1.1 Introduction to earthquakes

An earthquake occurs due to the sudden release of accumulated stress along a fault line, resulting in rapid movement known as fault slip. This movement can be described in terms of several key components:

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

1.2 Seismic and aseismic slips

1.3 Rate-and-state friction law

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

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

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

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

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

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

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

One common such law is called aging law

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

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

At steady state, the law is purely rate dependent

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

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

conditional stability. For a crack with size L embedded in an elastic medium with shear modulus G, the condition for unstable slip is

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

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

Nordström and Eriksson (2010)

CHAPTER II

METHODOLOGY

2.1 Numerical methods

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

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

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

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

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

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

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

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

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

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

2.2 SBP-SAT methods

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.2.0.3 SAT Penalty Terms.

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

2.3 Multigrid Method

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

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

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

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

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

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

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

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

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

CHAPTER III

SCIENTIFIC COMPUTING LIBRARIES AND LANGUAGES

3.1 PETSc

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

Key features of PETSc include:

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

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

    * Algebraic objects

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

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

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

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

3.2 AmgX

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

Key features of AMGX include:

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

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

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

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

3.3 HYPRE

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

Key features of PETSc include:

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

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

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

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

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

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

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

3.4 Review of several languages for scientific computing

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

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

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

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

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

3.4.2 C and C++.

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

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

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

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

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

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

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

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

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

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

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

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

3.5 Julia langauge

CHAPTER IV

PROBLEMS AND FORMULATIONS

CHAPTER V

RESULTS

5.1 Chapter Three Section One

5.1.1 Chapter three section one sub-section one.

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

CHAPTER VI

CONCLUSION

6.1 Chapter Four Section One

6.1.1 Chapter four section one sub-section one.

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

This is a sample citation: Schwartz (2012).

REFERENCES CITED

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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