STRUCTURE-PRESERVING HIGH PERFORMANCE COMPUTATIONAL METHODS FOR TRANSPORT IN POROUS MEDIA

A Dissertation

Presented to

the Faculty of the Department of Civil and Environmental Engineering

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

in Civil Engineering

by

Justin Chang

May 2017

STRUCTURE-PRESERVING HIGH PERFORMANCE COMPUTATIONAL METHODS FOR TRANSPORT IN POROUS MEDIA

Justin Chang

Approved:

Chair of the Committee
Kalyana Nakshatrala, Assistant Professor
Civil & Environmental Engineering

Committee Members:

Kaspar Willam, Distinguished Professor
Civil & Environmental Engineering

Cumaraswamy Vipulanandan, Professor
Civil & Environmental Engineering

Gino Lim, Professor
Industrial Engineering

Lennart Johnsson, Distinguished Professor
Computer Science

Matthew Knepley, Assistant Professor
Computational and Applied Mathematics
Rice University

Suresh K. Khator, Associate Dean
Cullen College of Engineering

Roberto Ballarini, Professor and Chair
Civil and Environmental Engineering

Acknowledgements

First and foremost, I would like to thank my advisor, Professor Kalyana Babu Nakshatrala, for his excellent guidance and mentorship throughout the last six years of my graduate studies. Your research interests and teaching methodologies have inspired me to make a truly life changing decision by going into the field of computational mechanics. It truly has been a memorable experience conducting research at the Computational and Applied Mechanics Laboratory (CAML).

I would also like to thank Professors Lennart Johnsson and Matthew Knepley for the many hours of fruitful and thought provoking discussions on topics related to high performance computing. I would also like to thank both my SCGSR mentor, Dr. Satisk Karra, and my former CAML office mate, Dr. Maruti Mudunuru, over at the Los Alamos National Laboratory (LANL) for exposing me to the DOE National Laboratory system as well as their helpful guidance towards my research. Much of the work presented in this dissertation would not have been possible without the many world-class computational scientists and researchers perusing through the PETSc, Firedrake Project, and PAPI mailing lists. Pariticipation in these e-mail threads has helped me not only run my software and computational frameworks more efficiently but also become a skilled scientific programmer.

Thank you, University of Houston, for the amazing facilities, resources, and faculty. The all-you-can-eat-buffet at Fresh Food Company is arguably the best buffet I have ever eaten at, and it has kept me well fed these last six years. I will miss this place dearly. I also want to acknowledge the research faculty over at the Center for Advanced Computing and Data Systems (CACDS) for providing the computational resources needed to test my computational frameworks. Thank you Dr. Jerry Ebalunode, Dr. Martin Huarte-Espinosa, Dr. Amit Amritkar, and other CACDS members for the very helpful emails and individual meetings.

I also want to acknowledge my mom, dad, and sister. I love y'all so much. My educational journey would not have been possible with their unrelenting love, support, and sacrifices these last 29 years. Life in Houston would not have been enjoyable without my friends and community over at the Houston Chinese Church. Thank you guys for the weekly words of encouragement and prayers. Last but not least, I would like to thank my Lord and Savior Jesus Christ because without Him, I am nothing.

STRUCTURE-PRESERVING HIGH PERFORMANCE COMPUTATIONAL METHODS FOR TRANSPORT IN POROUS MEDIA

An Abstract

of a

Dissertation

Presented to

the Faculty of the Department of Civil and Environmental Engineering

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

in Civil Engineering

by

Justin Chang

May 2017

Abstract

This dissertation aims at developing a novel and robust high performing parallel computational framework that can enhance the current predictive capabilities of numerical methods for large-scale subsurface transport applications. With increasing capacity and complexity of processors and memory systems, the need for improving the performance of subsurface flow and transport simulations has become an area of active research. Two of the most well known numerical deficiencies that the popular formulations and existing simulation packages suffer from are the inability to meet the non-negative constraint under anisotropic diffusion (i.e., they violate discrete maximum principles) and the inability of the standard finite element formulation to ensure local mass balance. Moreover, there is no platform-agnostic performance model that can simultaneously document both the hardware/architectural and algorithmic efficiencies of any numerical method or software package that is sensitive to memory-bandwidth limitations. Several existing parallel scientific libraries, such as the Portable and Extensible Toolkit for Scientific Computations (PETSc) and the Massively Parallel Reactive Flow and Transport (PFLOTRAN) libraries, have been developed to help predict subsurface phenomena and are used by many of today's leading hydrologists and geophysicists alike, but till date, the aforementioned concerns have not yet been resolved. Solutions to these numerical deficiencies have been proposed in literature but they do not address them concurrently in a high performance computing setting. Without a performance model, it is intractable to determine whether proposed modifications to these subsurface software packages would be fast, scalable, or efficient.

The objective of this dissertation is to present a computational framework that preserves important properties like local mass balance and positivity that can perform at a high level. Performance tools and methodologies are presented to guide users on how to understand the performance of such frameworks. This dissertation comprises of three sections. First, we present a conceptual performance spectrum that covers time-to-solution, arithmetic intensity, and equations solved per second for any parallel computational framework that solves partial differential equations. As proof of concept, this spectrum is utilized on a wide variety of state-of-the-art scientific libraries and multi-grid solvers like the FEniCS/Firedrake Projects, HYPRE, and Trilinos. It is shown that this spectrum can augment one's ability to understand both the hardware and algorithmic efficiencies of popular numerical techniques like the finite element method. Second, we propose an optimization-based computational framework that can ensure variationally consistent non-negative concentrations for large-scale anisotropic diffusion problems like Chromium remediation in the Sandia Canyon. The predicted computational performance of the proposed framework is based on a "perfect-cache" roofline model, loosely based on concepts originating from the performance spectrum, and it is shown that this roofline model can be used to predict how well the optimization-based framework can strong-scale. Third, we extend the proposed computational framework to solve advection-diffusion equations by employing the variational inequality solver. We also enforce local/element-wise mass conservation by discretizing the advection-diffusion equation using the Discontinuous Galerkin finite element method. Our numerical experiments demonstrate that the proposed variational inequality approach conserves local mass balance, ensures non-negative concentration fields, and can accurately model large-scale and non-linear coupled flow and transport phenomena such as the miscible displacement of oil in a heterogeneous reservoir.

Table of Contents

List of Figures

List of Tables

Notation

The following key abbreviations are used:

ABC	=	Arnold-Beltrami-Childress
AI	=	Arithmetic intensity
BLMVM	=	Bounded limited memory variable metric
CG	=	Conjugate Gradient method or Continuous Galerkin
CPU	=	Central processing unit
DG	=	Discontinuous Galerkin
FLOP	=	Floating-point operations
GAL	=	(Continuous) Galerkin
GMRES	=	Generalized minimal residual method
HPC	=	High performance computing
KSP	=	Krylov subspace solver
MCP	=	Mixed complementarity problem
MP	=	Minimization problem
QP	=	Quadratic programming
QP - TRON	=	Trust region Newton method
SNES	=	Scalable Newton equation solver
SP	=	Strong problem
SUPG	=	Streamlined Upwind Petrov-Galerkin
TBT	=	Total bytes transferred
VI	=	Variational Inequality
VI - SS	=	Semi-smooth method
VI - RS	=	Reduced-space active-set method
WF	=	Weak formulation

Chapter 1. Introduction

Subsurface energy sources present a significant and long-term opportunity for widespread power production. According to the U.S. Department of Energy (DOE), more than 80% of the total U.S. energy needs are satisfied by Engineered Subsurface Systems (ESS), Engineered Geothermal Systems (EGS), and conventional/non-conventional hydrothermal resources. These constitute nearly 100 Quadrillion BTU of energy required for domestic needs. Civil engineers, hydrologists, and geophysics alike have successfully developed technologies to work with complex systems that operate under various reservoir conditions. Such processes are often extremely large-scale so it is vital to also quickly and accurately model these processes.

The DOE has invested in the development of algorithms, methodologies, software and tools that can achieve this. For example, the Portable and Extensible Toolkit for Scientific Computations (PETSc [12] and the Massively Parallel Reactive Flow and Transport (PFLOTRAN [100]) simulator are just two of the many ongoing research endeavors useful for describing large-scale subsurface processes. However, there is still much room for improvement both numerically and computationally. It is well-known that many of the leading scientific software packages utilized today by leading computational scientists suffer from well-known deficiencies. Three of the most well-known ones are listed below.

1. **Local mass conservation**: The standard finite element method is notorious for not ensuring element-wise/local mass conservation. This important physical property is especially important for modeling subsurface reactive transport phenomena. Not ensuring this property at the element level creates numerical artifacts in the advected concentration fields as well as potentially violation maximum principles, which state that concentration values cannot be negative. Finite volume methods on the other hand are built upon conservation laws and

satisfy this property. PFLOTRAN employs the two-point flux finite volume scheme but this numerical method is highly inaccurate on unstructured grids and cannot model anisotropic porous mediums. Therefore, it is desireable to implement a robust finite element scheme that ensures local mass conservation and can incorporate anisotropic diffusivity.

2. **Non-negative concentrations**: When the porous medium is highly anosotropic, one may encounter numerical artifacts when using a non-monotone numerical discretization scheme. This can also violate maximum principles and positivity. Concentrations of chemical species physically cannot be negative, and this will potentially cause errors in the numerical simulation especially when nonlinear and reactive geochemical processes do occur. Standard numerical methods including the finite element and volume methods suffer from this setback, but one way to fix this problem is to employ optimization-based routines to enforce the non-negative constraint. Several studies over the years (e.g., [131] and the references within) have demonstrated the robustness of this approach, but none of them have extended this framework to large-scale realistic problems.

3. **Performance modeling** In this day and age of computing, both hardware and software are constantly evolving to cater to the needs of computational scientists needing to answer relevant scientific questions. Today's top supercomputers can operate at up to 93 peta-FLOPS per second (i.e., one quadrillion floating-point operations per second) while consuming significant power (15 mega Watts). It is important to understand how novel computational frameworks needed to enhance the current predictive capabilities of scientific software will behave on highly concurrent systems. Several benchmarks such as the memory benchmark STREAM [114] and the Roofline model [164] capture processor limitations and do not address algorithmic efficiency which is concerned with total work needed.

Moreover, complex PDEs require specific implementations of solvers and algorithms that may not be able to effectively utilize a particular hardware architecture. A performance model that can simultaneously model hardware, software, and algorithmic efficiencies of computational frameworks is highly desirable.

A computational framework that can simultaneously address these issues for large-scale subsurface flow and transport applications is still needed. A performance model aiding in the interpretation of how the computational framework for complex PDEs will behave on state-of-the-art computers is also still needed. It is important to not only obtain physically meaningful values for temperature and concentration of chemical species, but also to obtain them with high computational efficiency. *The objective of this dissertation is to present a robust framework that can enhance the current predictive capabilities of today's leading open-source software for modeling subsurface transport applications on high performance computing (HPC) systems.* The steps to achieving these outcomes are outlined in the next three chapters. Each chapter is self contained as they all have their own introduction, literature survey, and design methodology.

In Chapter 2, we present a framework called performance spectrum designed to help computational scientists understand how various flavors of today's software, algorithms, and computational platforms compare when solving partial differential equations (PDE). This chapter provides the reader with the knowledge to not only understand the performance and scalability of the proposed structure-preserving high performance computational framework for transport in porous media but also to gain insight into the performance and scalability of *any* PDE solver package. Particular emphasis is placed on comparisons between the standard finite element method and the locally conservative Discontinuous Galerkin finite element method. The performance spectrum framework augments the computational scientists' abilities to understand whether a scientific code is running efficiently.

In Chapter 3 we examine the anisotropic diffusion equation and propose a parallel optimization-based solver that can be used to enforce the discrete maximum principles and the non-negative constraint. Popular open-source scientific software packages like the PETSc and TAO are leveraged. The convex optimization solver utilized in this chapter not only demonstrates good algorithmic convergence on unstructured grids but also can model important and large-scale subsurface problems such as Chromium remediation in the Sandia Canyon. The large-scale performance of this proposed computational framework is justified using a performance model based on the concepts described by the performance spectrum.

In Chapter 4 we extend the proposed computational framework to ensure non-negative concentrations for advection-diffusion equations. Since the governing equation now has an asymmetric and non-self adjoint operator, the variational inequality approach, which is a more generalized optimization-based solver, is employed. Moreover, element-wise/local mass conservation is achieved under the finite element method using the Discontinuous Galerkin method. We study both the algorithmic and computational performance of the variational inequality approach for heterogeneous problems. The proposed non-negative computational framework is extended to non-linear coupled flow and transport problems like the miscible displacement of oil in a field-scale reservoir with randomly permeabiltiy, and the ramifications of not properly enforcing the non-negative constraint for such simulations is shown numerically.

Concluding remarks are outlined in Chapter 5. We also outline various ways the research presented in this dissertation can be extended.

Chapter 2. A performance spectrum for parallel computational frameworks that solve PDEs

2.1 INTRODUCTION AND MOTIVATION

Both efficient algorithms and software performing well on modern computing systems are crucial to address current scientific and engineering problems. These tools are important for bridging the gap between theory and real-world data. Such problems often need to tackle field-scale data sets using parallel computers, parallel algorithms, and programming tools such as OpenMP [41] and the Message Passing Interface (MPI) [64] and cannot be solved on a standard laptop or desktop. For example, hydrologists and geophysicists need to work with field-scale reservoirs which could span tens of kilometers and evolve on time scales of hundreds of years. Morever, such reservoir simulations involve complicated multi-phase and multi-component flows which require multiple complex equations to be solved accurately and efficiently. Atmospheric and climate modelers also require state-of-the-art techniques as both data assimilation and parameter estimation need to be performed quickly on meso-scale and global-scale applications. The US Department of Energy has invested in the development of several portable and extensible scientific software packages like PETSc [12, 11] and PFLOTRAN [99] that can help address such important large-scale problems. The time spent developing parallel computational frameworks is amortized when application scientists employ the packages in their work.

However, it is not always known whether the performance of a particular parallel computational framework or software will be satisfactory across a panoply of solvers and computing platforms. How can one really tell whether an algorithm is performing at its highest level? Is there room for improvement? Answering these questions in full is a Herculean task, but questions regarding the algorithmic and computational efficiency of scientific tools and libraries still need to be answered [82]. Hence, we need *performance models* which enable us to synthesize performance data into an understandable framework. Performance models can include many metrics of importance such as total floating point operations (FLOP), memory usage, inter/intra process/node communication, memory/cache bandwidth, and cache misses/hits. If not carefully optimized, some of the hardware resources can become unnecessary bottlenecks that result in costly and inefficient numerical simulations. Modern computer systems are quite complex and the performance can be difficult to predict with good accuracy. Conducting large-scale simulations on state-of-the-art supercomputers may require hundreds to thousands of hours of compute time, so it is highly desirable to have a performance model that can predict how a particular parallel computational framework may perform. The application or domain scientist may use software that either is not made in house or is a “black-box” tool, and it would be too time consuming, or impossible if source code is unavailable, to dissect the code and analyze the design of the subroutines and data structures. It is therefore desireable to analyze these codes as a whole.

2.1.1 Review of previous works

We now briefly highlight some useful approaches and models one could take to analyze and perhaps improve the performance of any parallel computational framework. One of the simplest measures one can utilize is the STREAM memory-bandwidth benchmark [114]. This benchmark measures sustainable memory-bandwidth on a single server and indicates the number of threads that saturates memory bandwidth. Memory-bandwidth is an important limitation to consider on modern machines [165, 115, 126]

The Roofline model [164, 106] captures peak achievable performance on a server taking into account both CPU and memory-bandwidth capabilities by introducing the

Arithmetic Intensity (AI). The AI is simply the measure of the total floating-point operations needed, total FLOP, over Total Bytes Transferred (TBT). Higher AI's indicate that the algorithm or computational framework is more computationally intensive and requires less bandwidth for a given amount of work. One is free to employ any cache model when determining the TBT metric for the roofline model. For example, scientists have developed a Sparse Matrix-Vector (SpMV) multiplications model [65] which is based on "perfect cache" (i.e., matrices and vectors are loaded and stored once from memory). SpMV is an integral part of iterative solvers for solving PDEs. It has been shown in [30] that the SpMV "perfect cache" model can also be used to accurately predict and understand the hardware performance of optimization-based solvers for enforcing discrete maximum principles. In [113], the authors employ matrix-free iterative methods for Stoke's equation, which is needed for lithospheric dynamic applications. The authors manually count the TBT based on source code. The advantage of matrix-free methods is that the sparse matrix-vector multiplication, which is memory-bandwidth limited, is not explicitly stored thus bringing the computational frameworks' upper-bound limit of the roofline closer to the Theoretical Peak Performance (TPP) region. TBT can also be determined based on memory level traffic or cache misses. The same analysis can be carried out for manycore architectures, such as Nvidia GPUs and the Intel Xeon Phi "Knights Landing" (KNL), in [91, 90].

For a more thorough analysis of performance, advanced software tools such as the HPCToolkit [3] and OpenSpeedShop [150] are used by scientific software developers and application scientists alike. These tools provide in-depth performance analyses of scientific codes and can also be used to debug the codes. Both of these tools rely on PAPI [119] which use low level hardware counters for important metrics like FLOPS, total CPU (central processing unit) cycles, and cache misses. These tools have proven to be extremely useful for computational scientists in all areas of computational physics and can provide a good understanding of the hardware performance of any computational framework for solving PDEs.

2.1.2 Main contributions

In this chapter, we provide a simple and easy-to-use performance model that can be used in addition to the techniques and tools mentioned above. Our performance model, which we term a *performance spectrum* [24], takes into account time-to-solution, AI based on cache misses, and equations solved per second. This model is applicable to any level of a scientific code, whether it be the entire computational framework or only particular phases or functions such as mesh generation, assembly of a matrix, or the solver step. *It is important to note that this tool is not intended to replace any of the aforementioned performance tools or models but to simply augment one's ability to quickly understand and diagnose the performance from both the hardware, software, and algorithmic stand point.* The main contributions of this chapter can be enumerated as follows:

1. We outline common issues pertaining to performance, ways to identify them, and methods to address them.

2. We present a model called performance spectrum that provides an enhanced understanding of the performance and scalability of algorithms and software.

3. We demonstrate that the proposed model can be utilized on existing popular software packages and solvers.

4. We apply the model to a more complicated and nonlinear PDE and document the parallel performance of the computational framework across HPC machines.

5. We discuss some possible ways in which this performance spectrum model can be extended.

The rest of the chapter is organized as follows. In Section 2.2, we outline some of the key performance issues one may come across when solving PDEs and how to address some of them. In Section 2.3, we propose a model, performance spectrum, which captures three critical metrics useful for understanding performance and scalability. In Section 2.4, we demonstrate possible ways one could utilize the proposed model by systematically comparing commonly used finite element packages and solvers. In Section 2.5, we extend the model to simulate nonlinear hydrostatic ice sheet flow equations. In Section 2.6, we run the nonlinear hydrostatic ice sheet flow equations across multiple compute nodes and study the performance. Concluding remarks and possible extensions of this work are outlined in Section 2.7. All the notational conventions employed in this chapter are introduced as needed.

2.2 COMMON PERFORMANCE ISSUES

The performance of any scientific software or algorithm will depend on a myriad of factors. First and foremost, good performance depends on efficient and practical implementation of the code. Application and domain scientists may not be interested in the intricate details of the code framework that they did not design, but they must still be cognizant of important computational issues that may inhibit performance dramatically. We now briefly highlight some common performances issues computational scientist may come across in their line of work:

- **Core/memory bindings**: The simplest way to maximize parallel performance for MPI applications is to properly enforce MPI process and memory bindings. This is particularly important for memory bandwidth-limited applications because, on most CPU architectures, the aggregate core bandwidth exceeds the CPU bandwidth to memory and it is important to use the CPUs in a multi CPU server in a balanced way. Furthermore, if multiple users share a compute

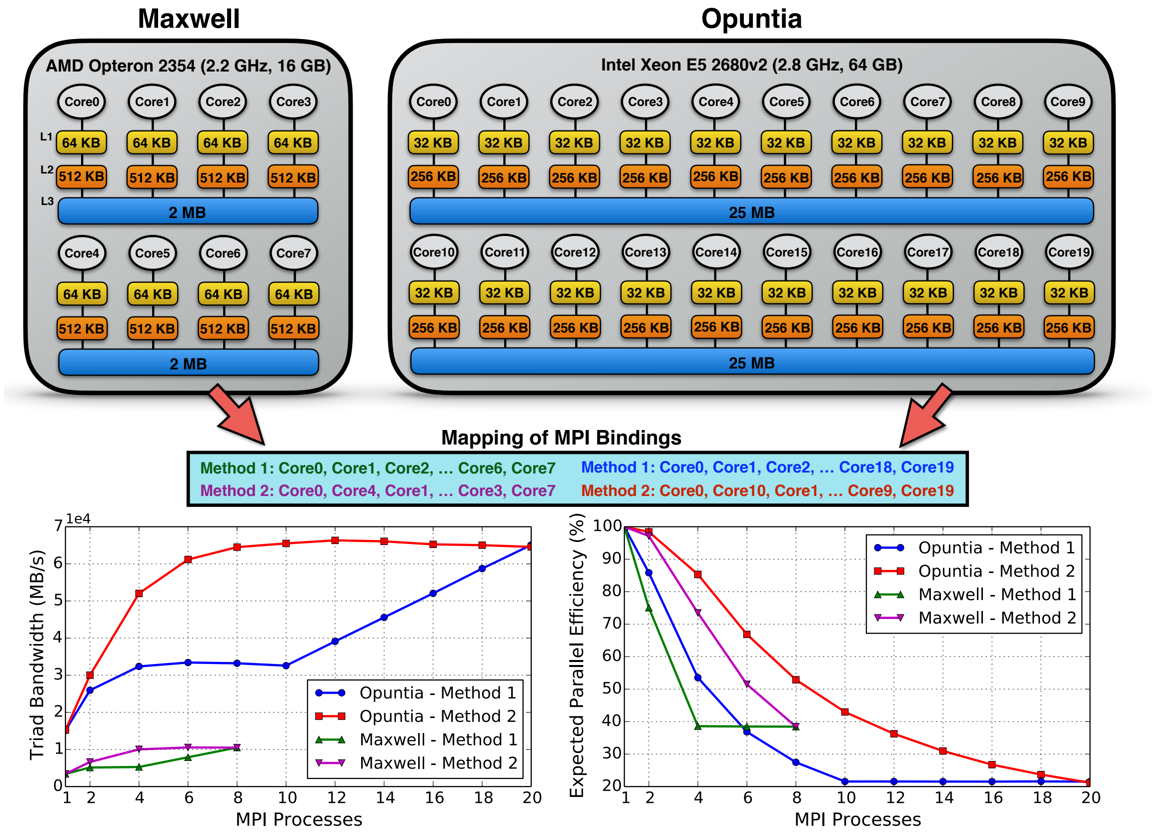


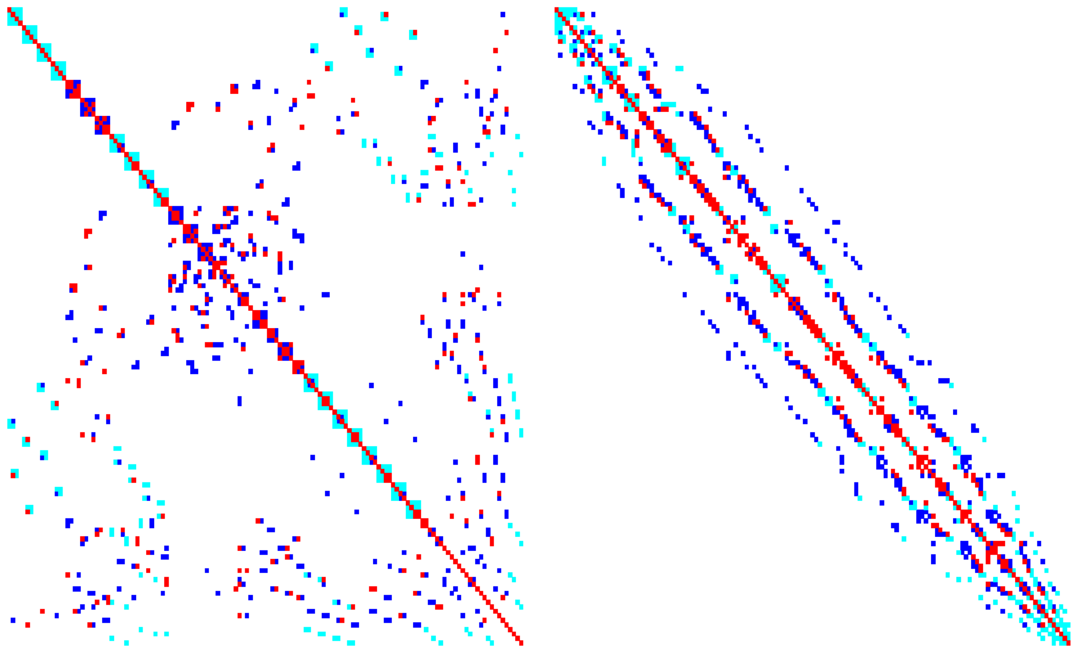
Figure 2.1: An overview of the STREAM measurement on two different compute nodes. The mapping of MPI bindings has a significant impact on the achievable memory bandwidth.

node, performance metrics can vary greatly as both memory resources and certain levels of cache are shared by others. Appropriate mapping methodologies for binding ranks to cores is vital for complex hardware architectures as well as for complex topological node layouts. Consider the single dual socket servers and their respective STREAM Triad benchmark results shown in Figure 2.1. Both the Maxwell and Opuntia servers possess two sockets where the physical cores are contiguously ordered. However, when the MPI processes are placed on alternating sockets, the achievable bandwidth is higher for a fixed number of cores by using the memory systems on both CPUs. For multi node performance, different binding techniques are required – memory references on a single node are several times faster than on a remote node. Process allocation must be carefully done so that communication across networks is minimized.

- **Hardware architecture**: The performance of any benchmark or software depends on the hardware architecture. In this chapter, we consider five different HPC systems with single node specifications listed in Table 2.1. It is evident from the STREAM Triad benchmark that different architectures have different levels of achievable memory-bandwidth. Some of the processors are recent (as of the writing of this dissertation), like the Intel KNL processor whereas others like AMD's "Barcelona" and Intel's "Ivybridge" processors are older. With increasing complexity of processors and memory systems, the challenge of good performance of solvers and algorithms has become an area of active research. A computational framework may solve a PDE efficiently on a laptop or small cluster, but that does not mean it will perform efficiently on a supercomputer. Understanding basic computer architectural concepts such as pipelining, instruction-level parallelism, and cache policies may offer excellent guidelines on how to speedup computations by several orders of magnitude. For example, Intel's KNL processor has two 512-bit vector units per core and may need fine-grained parallelism to fully exploit the 68 cores per CPU. If a code is not properly vectorized to utilize the 136 vector units capable of 16 floating-point operations per cycle or the 16 GB of onboard MCDRAM, it is possible that the speedup on this system will not be fully realized, and worse yet get outperformed by processors that have faster cores. Also, languages such as Python, which are used in some sophisticated finite element simulation packages, depend on the file system I/O because the interpreter executes system calls to locate the module and may need to open hundreds of thousands of files before the actual computation can begin. Designing and utilizing algorithms/languages/-compilers that are compatible with recent state-of-the-art HPC architectures is paramount [118], otherwise the computational performance may be exceedingly

Table 2.1: Single node specifications from each of the HPC systems used for this study. Note that Intel's "Knights Landing" processor has two different types of memory.

	Maxwell	**Opuntia**	**Edison**	**Cori**	**Cori**
Processor	AMD "Barcelona" Opteron 2354 2.2 GHz	Intel "Ivybridge" Xeon E5-2680v2 2.8 GHz	Intel "Ivybridge" Xeon E5-2695v2 2.4 GHz	Intel "Haswell" Xeon E5-2698v3 2.3 GHz	Intel "Knights Landing" Xeon Phi 7250 1.4 GHz
Year released	2008	2013	2013	2014	2016
Sockets	2	2	2	2	1
Cores/socket	4	10	12	16	68
Threads/core	1	1	2	2	4
L1 cache	8×64 KB 2-way associativity	20×32 KB 8-way associativity	24×64 KB 8-way associativity	32×64 KB 8-way associativity	68×64 KB 8-way associativity
L2 cache	8×512 KB 16-way associativity	20×256 KB 8-way associativity	24×256 KB 8-way associativity	32×256 KB 8-way associativity	34×1 MB 16-way associativity
L3 cache	2×2 MB 32-way associativity	2×25 MB 20-way associativity	2×30 MB 20-way associativity	2×40 MB 20-way associativity	- -
Memory type	DDR2-200 MHz	DDR3-1600 MHz	DDR3-1866 MHz	DDR4-2133 MHz	DDR4-2400 MHz, MCDRAM
Total Memory	16 GB	64 GB	64 GB	128 GB	96 GB (DDR4), 16 GB MCDRAM
Memory channels	4	8	8	8	6 (DDR4), 8 (MCDRAM)
Compiler used	GNU	GNU	Cray	Cray	Cray
STREAM Triad	10.5 GB/s	64.5 GB/s	102 GB/s	116 GB/s	90 GB/s (DDR4), 480 GB/s (MCDRAM)



(a) Default ordering **(b)** Optimized ordering

Figure 2.2: Assembled sparse matrix where red represents positive numbers, blue represents negative numbers, and cyan represents allocated but unused nonzero entries.

poor.

- **Domain decomposition**: The global ordering and partitioning of the computational elements in a parallel computing environment, particularly for problems with unstructured grids, affect both spatial and temporal cache locality. Consider the assembled sparse matrices shown in Figure 2.2. If the nonzero data entries are not properly grouped together, the code will invoke expensive cache misses and create little opportunity to use data in a cache line and reuse data in the cache. Consequently, this create serial bottlenecks at the cache/memory levels. Several mesh/graph partitioners such as Chaco [73], METIS/ParMETIS [80], and PTSCOTCH [34] are designed to optimize locality and balance the workload among MPI processes. Some graph partitioners use a simple model of communication in seeking to achieve load balance with minimum communication while others use a more detailed communication model to better capture the minimum communication needed by using hypergraphs instead of regular graphs as a basis for partitioning. Understanding which type of partitioning to use for the PDE problem at hand (e.g., spectral partitioning, geometric partitioning, multilevel graph partitioning, etc.) can significantly reduce the amount of communication and lead to higher efficiency and degree of concurrency.

- **Solver convergence**: Arguably one of the most important performance factors to consider for solving PDEs is the convergence rate of the solver. Direct methods like Gaussian elimination [63] as well as its sparse counterparts such as MUMPS [7] and SuperLU_DIST [98] can solve problems in parallel but may have huge memory requirements as the problem size is scaled up due to fill-in during factorization. Scalable and efficient solvers typically rely on the novel combination of iterative solvers and preconditioners. The Krylov Subspace (KSP) and Scalable Nonlinear Equations Solvers (SNES) features in the PETSc library coupled with robust preconditioners [156, 20] is a popular methodology for solving large and complex PDEs. Novel combinations and tuning of solver parameters provide powerful and robust frameworks that can accurately and quickly converge to a specified residual tolerance, even for complex coupled multi-physics problems [29, 25, 27]. Simple preconditioners such as Jacobi or Incomplete Lower Upper (ILU(0)) factorization may be fast for smaller problems, but the computational cost will soar because the number of solver iterations needed with Jacobi or ILU(0) will rapidly grow with problem size. Scaling up the problem under these choices of preconditioning will be extremely time consuming and may not even converge for larger or more complicated problems. Other more robust preconditioners, like the geometric and algebraic multigrid method, might have a more expensive setup time for smaller problems but have been demonstrated to maintain relatively uniform convergence for larger problems, even those that are nonsymmetric and indefinite [19, 1].

- **Serial vs parallel performance**: Suppose a computational framework employs proper MPI bindings, good utilization of hardware resources for a single MPI process, balanced mesh partitioning, and fast algorithmic solvers yet experiences little parallel speedup. What could be the cause of the lack of scaling in a serially efficient computational framework? *A code that is serially efficient may often translate to poor parallel efficiency.* The problem with parallel performance metrics like strong-scaling and weak-scaling is that they can easily be "gamed" to appear more efficient. For example, achieving near perfect parallel efficiency can be accomplished by making a code serially inefficient. If one chooses to forgo basic optimization steps such as loop unrolling and vectorization, the computation will be slower. One could also insert many embarassingly parallel yet unnecessary computations to make both the strong-scaling and FLOP rate metrics look good, but the overall time-to-solution is greatly increased. Amdahl's law [6] demonstrates that computationally efficient tasks will almost always experience poor speedup at high levels of concurrency because the fraction of the computation that can benefit from parallelism reduces.

These important performance issues should not be overlooked when analyzing the performance of a parallel computational framework. There are also several quick strategies for understanding and identifying bottlenecks on a specific HPC system. For example, it is well-known that SpMV is an operation that is sensitive to the memory-bandwidth. These operations have very low AI's which can present itself as a bottleneck at the memory level on a single node. A simple test one can perform is to run the SpMV operation in parallel, and if it does not scale well on a single server in the strong sense, the memory-bandwidth is clearly limiting the performance. One can confirm this by running some simple vector operations like the vector sum and scalar multiplication to see if they experience the same scaling issues. In addition, one can test the vector dot product operation in order to detect problems with the network interconnect or memory latency issues. The PETSc performance summary [12] provides comprehensive insight into the performance of many of these important operations including load balancing. The summary also provides information on the functions consuming most of the time. However, not all scientific software have readily available performance summaries, so a performance model amenable to any code implementation is needed to help answer common performance questions.

2.3 PROPOSED PERFORMANCE SPECTRUM

The general concept of the performance spectrum model is illustrated by Figure 2.3. This model is designed to simultaneously capture both the hardware/architectural exploitation as well as the algorithmic scalability of a particular parallel computational framework. First and foremost, we need the time-to-solution since this is the metric of most importance to application scientists needing to execute large-scale simulations on state-of-the-art HPC systems. One may optionally document the total number of solver iterations needed for convergence. However, simply knowing the wall-clock time a computational framework needs to perform a task tells us little about the computational and algorithmic efficiency. In order to understand how fast (or slow) a simulation is, we need to introduce two more metrics.

2.3.1 Arithmetic Intensity

The second metric of interest is what we refer to as *Arithmetic Intensity (AI)*. As described in [164], the AI of an algorithm or software is a measure that aids in estimating how efficiently the hardware resources and capabilities can be utilized. For the five machines listed in Table 2.1, it is well-known that the limiting factor of performance for many applications is the memory-bandwidth. Thus, codes that

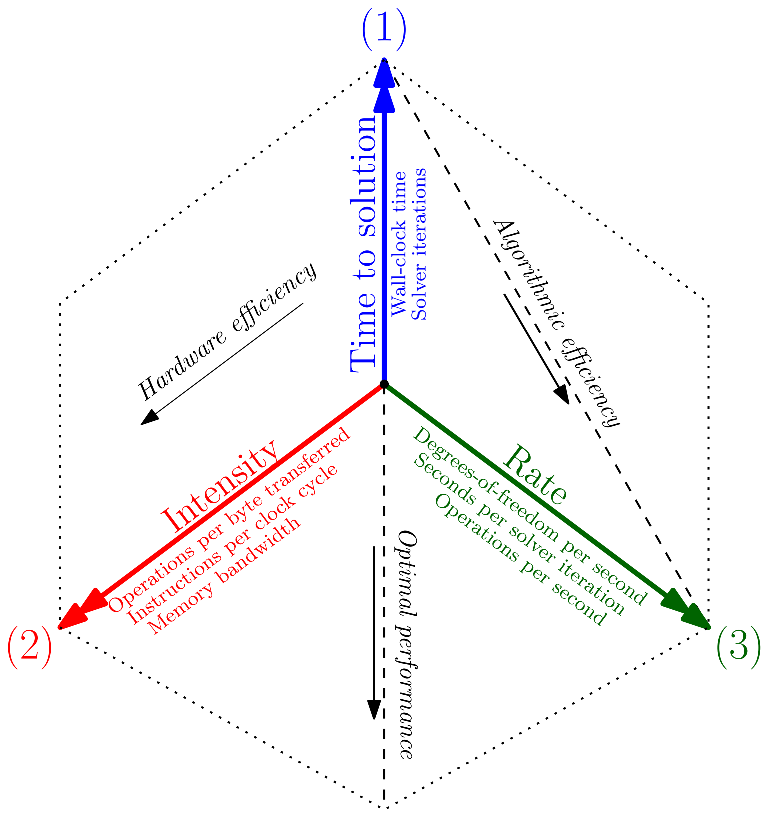


Figure 2.3: Proposed performance spectrum that documents time, intensity and rate. Intensity is defined as arithmetic intensity (FLOP to TBT ratio) based on cache misses, and rate is defined as degrees-of-freedom solved per second.

have a high AI have a possibility of reusing data in cache and have lower memory bandwidth demands. It should be noted, however, that performance depends on many factors such as network latency and file system bandwidth, and the arithmetic intensity alone cannot be used to predict performance.

The general formula for the AI is defined as

$$\text{AI} := \frac{[\text{Work}]}{[\text{TBT}]}, \tag{2.3.1}$$

where [Work] is the total amount of computational effort, typically what one would refer to as FLOPs. The [TBT] metric is a measure of data movement between the core/CPU and memory. A cache model is needed in order to not only determine the TBT but also to understand what amount of useful bandwidth is sustained for a given cache line transfer. One can employ any cache model for this purpose, such as perfect cache, total number of load and store instructions at the core level, traffic at the memory level, or data cache misses. Different cache models are useful for interpreting different behavioral trends, and the choice of cache model depends on the application or research problem at hand. In this chapter, we base [TBT] on the total number of cache misses and cache line size. The formula for obtaining the TBT for the L1, L2, and L3 cache levels is expressed as

$$\mathrm{TBT}_{\mathrm{Lx}} = [\mathrm{Lx\ misses}] \times [\mathrm{Lx\ line\ size\ (byte)}]. \tag{2.3.2}$$

The simplest way to define [Work] is as the total number of floating-point operations, denoted FLOPs. Thus the AI based on Lx cache misses is formally written as

$$\mathrm{AI}_{\mathrm{Lx}} = \frac{[\mathrm{FLOPs}]}{\mathrm{TBT}_{\mathrm{Lx}}}. \tag{2.3.3}$$

If a solver or algorithm experiences a large number of cache misses at the last level, memory may impede performance.

Sometimes the exact TBT of a particular algorithm is not of interest. Instead, an application scientist may only care about the relative measure, i.e., whether the AI is higher or lower compared to either another algorithm, a different implementation of the same algorithm, or a different processor. Thus, one may simply look at the ratio of FLOPS and cache misses. Equation (2.3.3) may be simplified to

$$\mathrm{AI}_{\mathrm{Lx}} = \frac{[\mathrm{FLOPs}]}{[\mathrm{Lx\ misses}]}. \tag{2.3.4}$$

Every machine listed in Table 2.1 has a cache line size of 64 bytes for all levels of cache. Different CPUs may have different line sizes and hence a cache miss may imply different memory demands on different processor architectures. The remainder of the chapter shall refer to the above formula for estimating the intensity metric.

Remark 2.3.1. *It should be noted that PAPI's methodology for counting FLOPS may be highly inaccurate for the "Ivybridge" systems listed in Table 2.1. The hardware counters only count the instructions issued and not the ones executed or retired. This is paramount for iterative solvers that rely on SpMV operations because as the codes spend time waiting for data to be available from memory, they will reissue the floating-point instructions multiple times. These reissues, coupled with incomplete filling of a vector unit instruction, can lead to overcount factors of up to 10 times. For a more thorough discussion on the issue of overcounts, see [162] and the references within. However, PAPI's FLOP counters are disabled on both of Cori's Intel processors due to the aforementioned issues, so if a software developer is really interested in approximating the FLOP count of a particular code, they could insert counting mechanisms into the code. PETSc provides an interface and guidelines for manual FLOP counting, and thus FLOP counts for computational frameworks using it can be obtained through the performance summary output.*

Remark 2.3.2. *The correlation between AI and speedup on a single node may not always hold true in a cluster sense (i.e., scaling when communication networks are involved). The mechanisms used for MPI process info exchanged is very different when the processes are on the same node as opposed to on different nodes. An application scientist must be fully cognizant of not only the HPC processor specification but also the network topology as well as the interconnect bandwidth and latency.*

2.3.2 Rate

Although $\mathrm{AI}_{\mathrm{Lx}}$ is useful for comparatively estimating the performance a particular parallel framework may attain, it does not necessarily aid in predictions of time-to-solution. Consequently, this means that AI can also easily be "gamed" to appear high but the code consumes large amounts of wall-clock time. For example, small computationally intensive routines such as DGEMM [45] can be inserted to artificially inflate the computational demands and increase the AI. Other performance models, such as the Roofline model, would indicate that this is a favorable trend while ignoring the fact that more time is spent than necessary. This is also why the traditional FLOP rate metric, which can also easily be gamed as discussed in Section 2.2, is not helpful either. Instead of measuring the total FLOPS executed per second, we measure the total degrees-of-freedom solved per second, hence the *Rate* metric needed to complete the performance spectrum is defined as

$$\text{Rate}_1 := \frac{[\text{DOFs}]}{[\text{total time (seconds)}]}, \tag{2.3.5}$$

where [DOFs] simply refers to the total number of degrees-of-freedom or discrete component-wise equations that need to be solved.

Definition 2.3.3 (Static-scaling)**.** *Equation* (2.3.5) *is an integral component of what we refer to as* static-scaling*, where we increase the problem size but fix the concurrency. This is a complete reversal to the classical definition of strong-scaling where we fix the problem size but increase the concurrency. Static-scaling plots time-to-solution versus the total degrees-of-freedom solved per second for a variety of problem sizes, so it also has characteristics similar to the classical definition of weak-scaling where both problem size and concurrency is increased.*

Figure 2.4 contains a pictorial description of a static-scaling plot and illustrates how to visually interpret the data points. A scalable algorithm is $\mathcal{O}(n)$ where $n :=$ [DOFs] is linearly proportional to [total time (seconds)], so it is desirable to see a PDE solver maintain a constant rate metric for a wide range of problem sizes. The behavior of parallel computational frameworks for solving PDEs is not simple because 1) problems too small for a given MPI concurrency experience large communication to computation ratios (hence strong-scaling effects) and 2) large problems may have

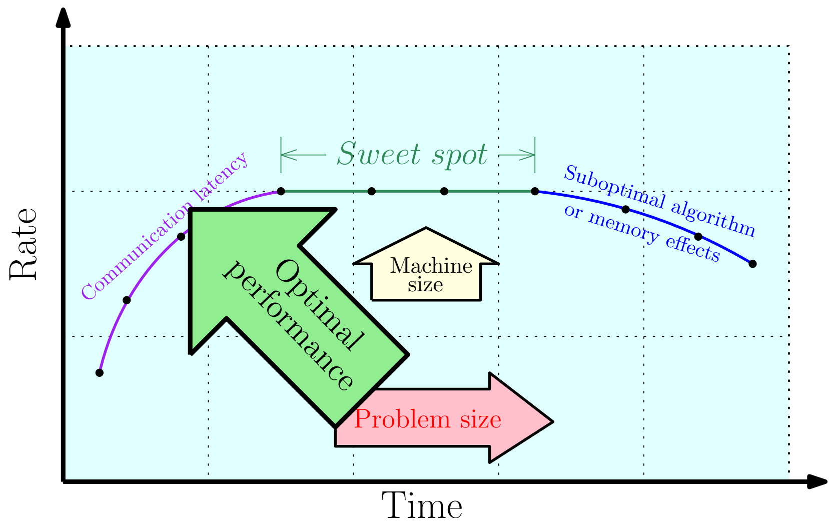


Figure 2.4: Static-scaling plot. By fixing the MPI concurrency and increasing the problem size, the rate axis is the degrees-of-freedom solved per second. Algorithmic efficiency is achieved when a flat line is observed as the problem is scaled up.

infavorable memory accesses. The static-scaling plots are designed to capture both strong-scaling and weak-scaling characteristics and can give a good indicator of the ideal range of problem sizes for a given MPI concurrency.

The tailing off to the right of the static-scaling plot has two potential reasons. First, problem size affects how memory is allocated and accessed. Larger problem sizes may see an increase in memory contention as well as affect the access pattern to main memory. Thus more time is spent waiting on data as opposed to performing calculations. However, another reason the tailing off occurs is because solvers for complex PDEs or computational domains may not always be $\mathcal{O}(n)$. Suboptimal algorithmic convergence may maintain a consistent level of hardware utilization but require more iterations and FLOPs. To determine whether suboptimal algorithmic convergence plays a role in the deterioration of the static-scaling plot, equation (2.3.5) can be modified as

$$\text{Rate}_2 := \frac{[\text{DOFs}]}{[\text{time (seconds)}] \times [\text{no. of solver iterations}]}. \tag{2.3.6}$$

This equation averages out increases in time due to an increase in iteration count. If a flat line is observed using this metric, then poor algorithmic scalability did have a negative impact on the static-scaling results.

Alternatively, if one is more interested in the performance gain for each MPI process, equation (2.3.5) can also be modified into

$$\text{Rate}_3 := \frac{[\text{DOFs}]}{[\text{time (seconds)}] \times [\text{no. of MPI processes}]}. \tag{2.3.7}$$

This metric presents the average degrees-of-freedom solver per second for each MPI process or core utilized.

2.3.3 Using the performance spectrum

The arithmetic intensity and static-scaling components of the spectrum offer a variety of strategies for interpreting the performance and scalability of any computational framework. Good performance is achieved when a computational framework achieves low time-to-solution, high arithmetic intensity, and flat static-scaling lines. The theoretical peak rate of degrees-of-freedom solved per second could be unknown for a particular algorithm, but the intensity metric can help us understand whether the static-scaling lines are good by estimating how well it is efficently using the available hardware resources. We outline three possible ways one could use the performance spectrum model:

1. Hardware limitations: As mentioned in Section 2.2, the hardware configuration of the compute nodes plays a vital role in the performance spectrum because different systems have different core counts, frequencies, and memory architectures. Understanding how PDE solvers behave on different systems is vital for disseminating software to the scientific and HPC communities. The different cache sizes listed in Table 2.1 will be reflected in equation (2.3.5). AI_{LX} is likely

to differ on different processors due to possible differences in cache sizes and cache policies. Furthermore, different processors have different clock frequencies, arithmetic capabilities, and memory bandwidth. Moreover, various GNU, Intel, and Cray compilers generate different executables that also depend on optimization flags used. Compiled code also depend on the data structures used as well as code constructs. A particular platform may be better suited for certain PDE applications. The performance spectrum model is useful for quickly visualizing and comparing the impact of platform characteristics, software, compiler options, and algorithms.

2. Software/solver implementation: There are several software packages suited for sophisticated finite element simulations such as the C++ based DEAL.II package [14], the Python based Firedrake Project [141], the Python/C++ based FEniCS Project [4], the C++ based LibMesh [86], and MOOSE[55] projects. These scientific libraries all use PETSc's linear algebra backend, but they can also use other packages such as HYPRE [51] and Trilinos/ML [74]. How well can specific solvers or software packages solve the same boundary value problem? Algebraic multigrid solvers have various theoretical approaches and implementation strategies, so it is entirely possible that certain solver configurations are better suited for a particular hardware architecture or PDE. Multigrid solvers for optimization remain a difficult research problem, but will be imperative for sustaining a high level of computational performance. Quick visual representations of the AI and equations solved per second can certainly guide programmers and scientists in the right direction when designing or implementing different software and solvers.

3. Numerical discretization: Finally, various flavors of numerical discretizations such as the finite difference, finite element, and finite volume methods not only have

different orders of mathematical accuracy but different number of discrete equations to solve for a given mesh. Consider the Continuous Galerkin (CG) and Discontinuous Galerkin (DG) finite element methods – clearly the DG method has more degrees-of-freedom since each element has its own copy of a geometric node, but does that necessarily mean it is more time consuming? For example, if the CG and DG elements each take roughly T seconds to attain a solution for the same computational domain, then the latter element clearly has a higher rate metric because it has more degrees-of-freedom for a given h-size, hence a bigger numerator in equation 2.3.1. This is important for computational scientists and mathematicians that want to compare the convergence rate of various numerical methods particularly if p-refinement studies are involved. A cost benefit analysis can be performed when comparing the numerical accuracy vs computational cost, often quantified using a work-precision diagram [87]. One could also compare the impact finite element discretizations have on different geometric elements (e.g., tetrahedrons, hexahedrons, wedges, etc.). The performance of any numerical method depends on the hardware limitations and software implementations, but this spectrum can be useful for comparing different and available discretizations and polynomial orders in sophisticated finite element simulation packages.

2.4 DEMONSTRATION OF THE PERFORMANCE SPECTRUM

As proof-of-concept, we apply the proposed performance spectrum to study the computational performance of a couple of popular finite element packages when used to solve the steady-state diffusion equation. A series of demonstrations shall enrich our current understanding of how hardware limitations, software implementation, numerical discretization, and material properties can impact the performance and scalability. We restrict our studies to the C++ implementation of the FEniCS Project and the Python implementation of the Firedrake Project, both of which leverage several scientific libraries and solvers such as PETSc, HYPRE, and Trilinos/ML solvers. The GMRES iterative solver is used with various algebraic multigrid solvers set to a relative convergence tolerance of 10^{-7}. All numerical simulations are performed on a single Maxwell (AMD Opteron 2354) and Opuntia (Intel Xeon E5-2680v2) node as described in Table 2.1. In this section, the performance spectrum model is used only to assess the assembly and solve steps.

The steady-diffusion equation gives rise to a second-order elliptic partial differential equation. To this end, let Ω denote the computational domain, and let $\partial\Omega$ denote its boundary. A spatial point is denoted by $\mathbf{x}$. The unit outward normal to the boundary is denoted by $\widehat{\mathbf{n}}(\mathbf{x})$. The boundary is divided into two parts: Γ^{D} and Γ^{N}. The part of the boundary on which Dirichlet boundary conditions are prescribed is denoted by Γ^{D}, and the part of the boundary on which Neumann boundary conditions are prescribed is denoted by Γ^{N}. For mathematical well-posedness we assume that

$$\Gamma^{\mathrm{D}} \cup \Gamma^{\mathrm{N}} = \partial\Omega \quad \text{and} \quad \Gamma^{\mathrm{D}} \cap \Gamma^{\mathrm{N}} = \emptyset. \tag{2.4.1}$$

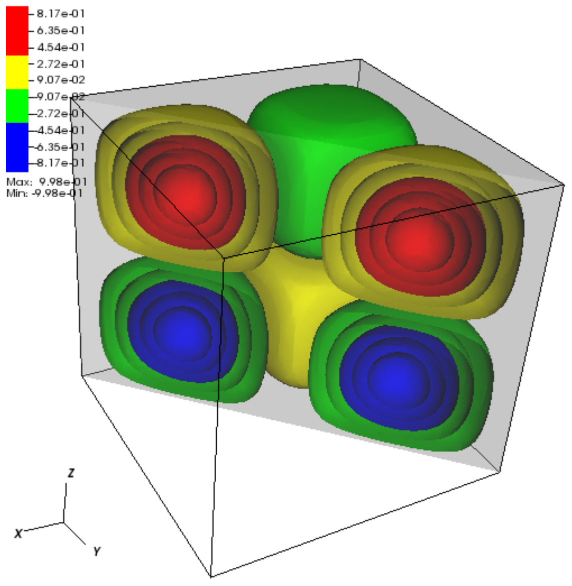
The corresponding boundary value problem takes the following form

$$-\mathrm{div}[\mathbf{D}(\mathbf{x})\mathrm{grad}[c(\mathbf{x})]] = f(\mathbf{x}) \qquad \text{in } \Omega, \tag{2.4.2a}$$

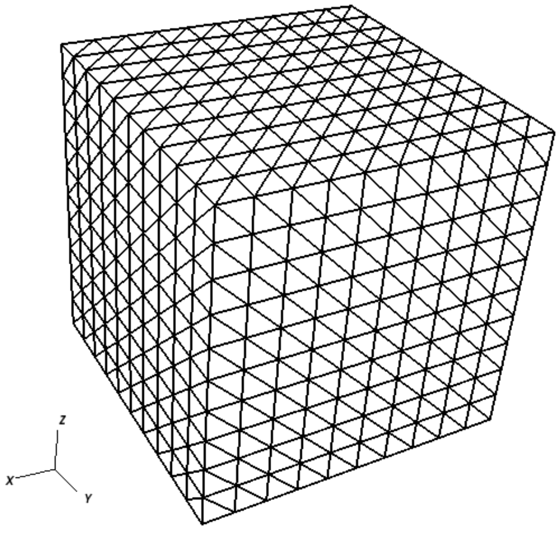
$$c(\mathbf{x}) = c^{\mathrm{p}}(\mathbf{x}) \qquad \text{on } \Gamma^{\mathrm{D}}, \quad \text{and} \tag{2.4.2b}$$

$$-\widehat{\mathbf{n}}(\mathbf{x}) \cdot \mathbf{D}(\mathbf{x})\mathrm{grad}[c(\mathbf{x})] = q^{\mathrm{p}}(\mathbf{x}) \quad \text{on } \Gamma^{\mathrm{N}}, \tag{2.4.2c}$$

where $c(\mathbf{x})$ is the scalar concentration field, $\mathbf{D}(\mathbf{x})$ is the diffusivity coefficient, $f(\mathbf{x})$ is the volumetric source, $c^{\mathrm{p}}(\mathbf{x})$ is the prescribed concentration on the boundary, and $q^{\mathrm{p}}(\mathbf{x})$ is the prescribed flux on the boundary. Assuming $\mathbf{D}(\mathbf{x}) = \mathbf{I}$, we consider the



(a) Analytical solution



(b) Mesh

Figure 2.5: Analytical solution of the steady-state diffusion example and the corresponding mesh skeleton of the structure grid containing tetrahedrons.

Table 2.2: Mesh discretization and CG1 L_2 error norm with respect to h-refinement.

h-size	Tetrahedrons	Vertices	FEniCS L_2 error	Firedrake L_2 error
1/20	48,000	9,261	1.48E-02	2.96E-02
1/40	384,000	68,921	3.90E-03	7.77E-03
1/60	1,296,000	226,981	1.75E-03	3.51E-03
1/80	3,072,000	531,441	9.89E-04	1.99E-03
1/100	6,000,000	1,010,301	6.34E-04	1.28E-03
1/120	10,368,000	1,771,561	4.41E-04	8.88E-04
1/140	16,464,000	2,803,221	3.24E-04	6.52E-04
			slope: 1.97	slope: 1.96

following analytical solution and corresponding forcing function on a unit cube:

$$c(\mathbf{x}) = \sin(2\pi x)\sin(2\pi y)\sin(2\pi z) \quad \text{and} \tag{2.4.3}$$

$$f(\mathbf{x}) = 12\pi^2\sin(2\pi x)\sin(2\pi y)\sin(2\pi z). \tag{2.4.4}$$

Homogeneous Dirichlet boundary conditions are applied on all faces, and the analytical solution for $c(\mathbf{x})$ is presented in Figure 2.5. These next few studies shall consider the following h-sizes on a structured tetrahedron mesh: 1/20, 1/40, 1/60, 1/80, 1/100,

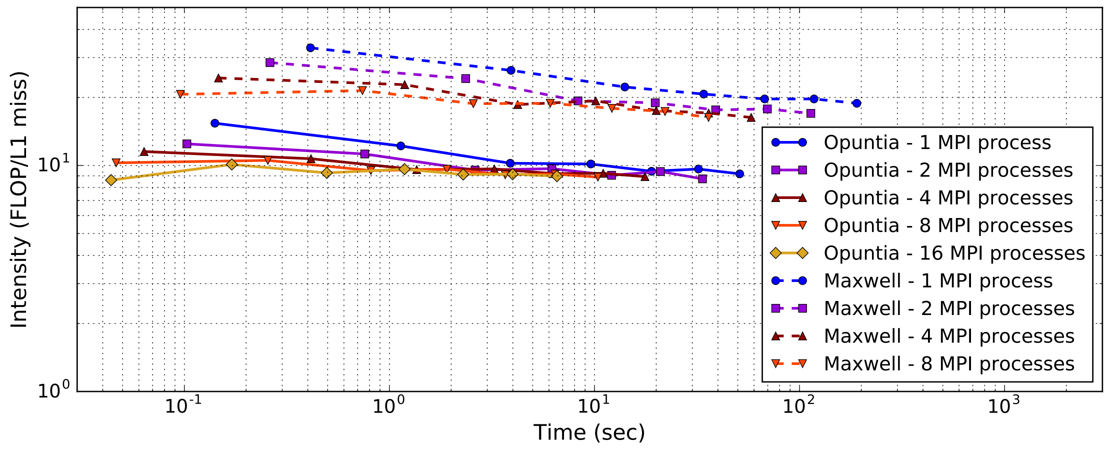


Figure 2.6: Demo #1: L1 arithmetic intensity for the FEniCS finite element package with PETSc's algebraic multigrid solver on a single Opuntia (Intel Xeon E5-2680v2) and Maxwell (AMD Opteron 2354) compute node.

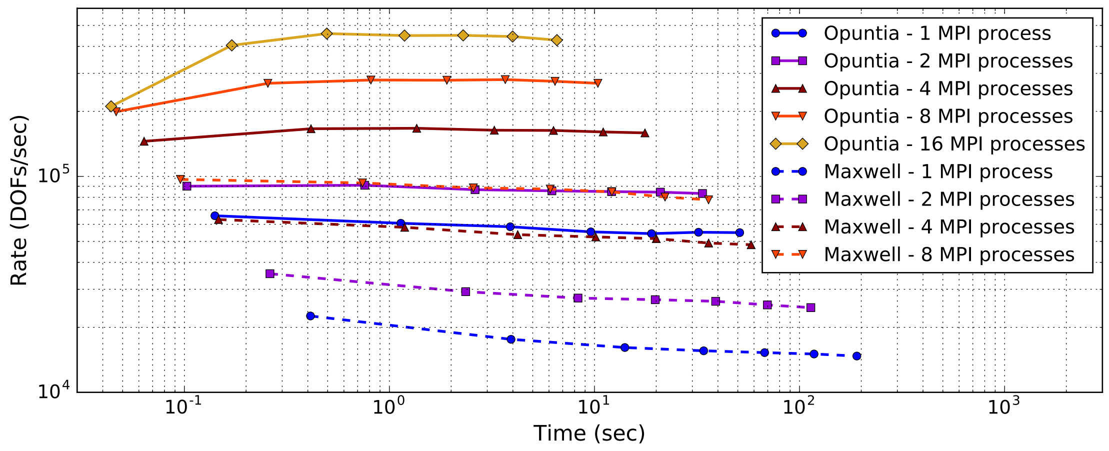


Figure 2.7: Demo #1: Static-scaling for the FEniCS finite element package with PETSc's algebraic multigrid solver on a single Opuntia (Intel Xeon E5-2680v2) and Maxwell (AMD Opteron 2354) compute node.

1/120, and 1/140. All mesh information and L_2 error norms with respect to the FEniCS and Firedrake implementations of the continuous Galerkin (CG1) element is listed in Table 2.2.

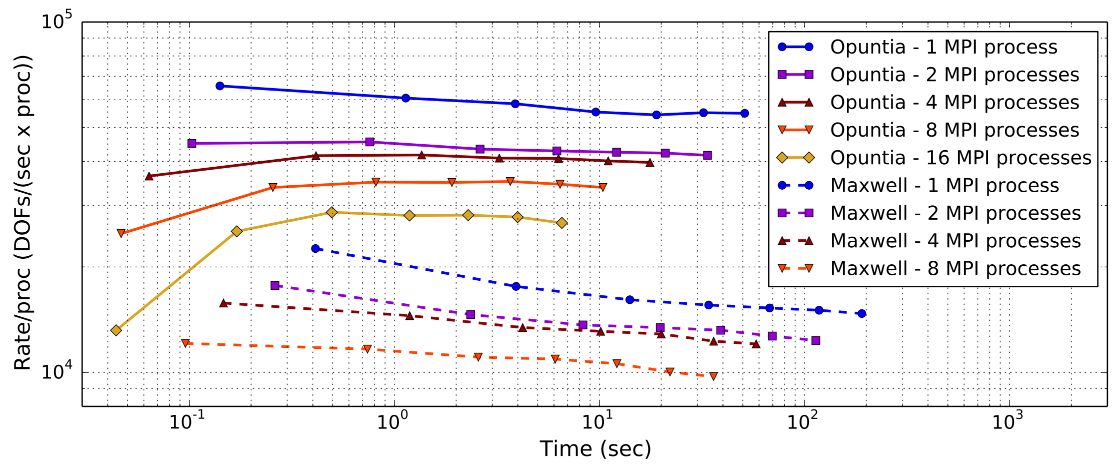


Figure 2.8: Demo #1: Static-scaling per MPI process for the FEniCS finite element package with PETSc's algebraic multigrid solver on a single Opuntia (Intel Xeon E5-2680v2) and Maxwell (AMD Opteron 2354) compute node.

2.4.1 Demo #1: AMD Opteron 2354 vs Intel Xeon E5-2680v2

We first compare the AI betweeen a single Intel Xeon E5-2680v2 (Opuntia) and AMD Opteron 2354 (Maxwell) compute node for FEniCS's implementation of the CG1 element coupled with PETSc's algebraic multigrid preconditioner. The AI_{L1}, as seen from Figure 2.6, gradually decreases with mesh refinement. Moreover, increasing the number of MPI processes also reduces the AI. The Intel processor has smaller L1 and L2 caches compared to the AMD processor, which explains why the former processor has lower AIs. It can be concluded that a higher AI on a different machine does not necessarily translate to better performance because clock rates and memory bandwidths differ. The fact that differences in AI_{L1} does not directly relate to time-to-solution can be seen in Figure 2.6.

The static-scaling plot is shown in Figure 2.7. It is clear that the Intel processor is capable of solving more degrees of freedom per second than the AMD processor. Increasing the number of MPI processes improves the Rate_1 metric, which is expected since time to solution is amortized. Employing Rate_3 from equation (2.3.7), as seen in Figure 2.8 gives us a better insight into the effect adding more MPI processes

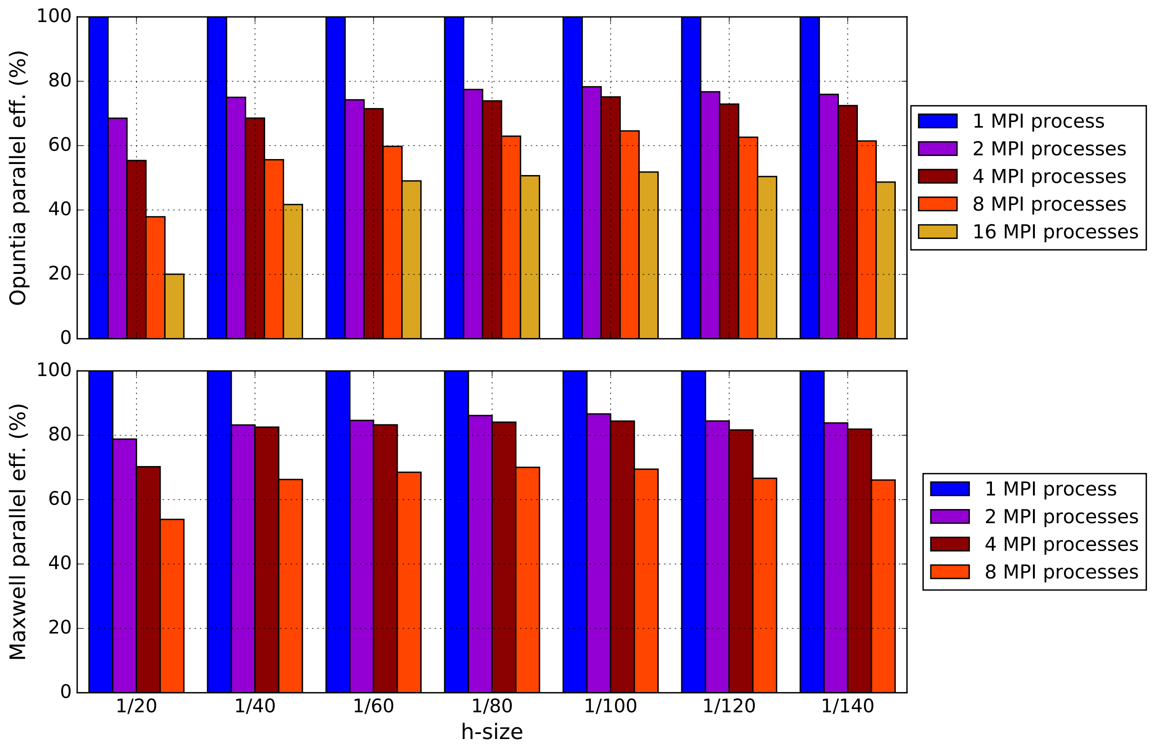


Figure 2.9: Demo #1: Strong-scaling efficiency for the FEniCS finite element package with PETSc's algebraic multigrid solver on a single Opuntia (Intel Xeon E5-2680v2) and Maxwell (AMD Opteron 2354) compute node.

onto a single node has on the static-scaling performance. We also note that when only one or two MPI processes are used, the degrees-of-freedom solved per second metric degrades as the problem size increases. We also observe that the line plots for Intel reach higher apexes as more MPI processes and larger problems are solved. The lines curves "dipping" to the left indicate a degradation in parallel performance – the problems are very small (e.g., h-size of 1/20 resulting in 9,261 degrees-of-freedom distributed among 16 MPI processes means each process solves roughly only 580 equations) thus more of the execution time is spent on interprocess communication and latencies than actual computation. Both the Rate$_1$ and Rate$_3$ lines decrease with problem size on the AMD node, whereas the line plots for the Intel node are relatively flat, suggesting that the FEniCS and PETSc combination is in fact an algorithmically scalable combination for the problem at hand.

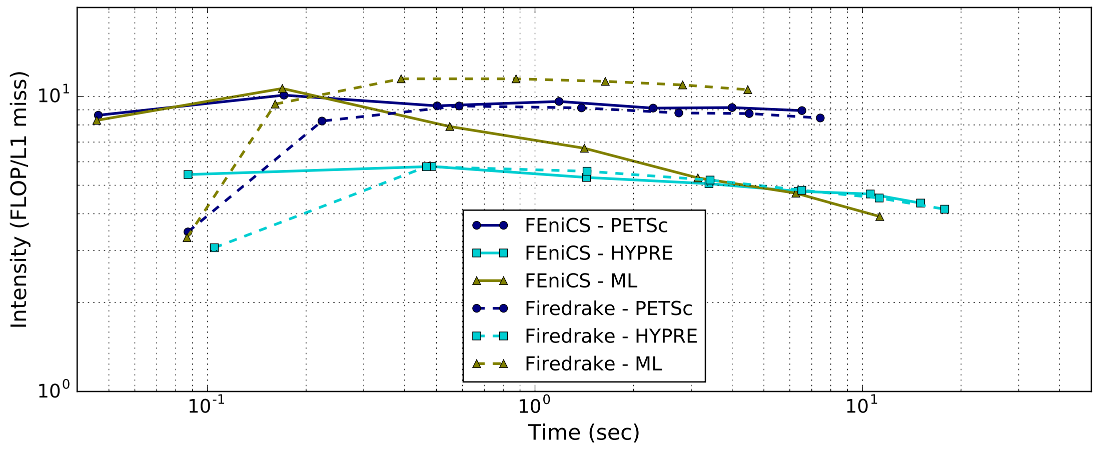


Figure 2.10: Demo #2: L1 arithmetic intensities for the FEniCS and Firedrake finite element packages with various solver packages on a single Opuntia (Intel Xeon E5-2680v2) node with 16 MPI processes.

Figure 2.9 depicts the parallel speedup on the two different nodes. The parallel performance for the smaller h-sizes is significantly worse due to the lack of computation needed for a given MPI concurrency. It is interesting to note that the speedup on the AMD node is slightly greater than on the Intel node. Recalling from Figure 2.6 that the $\mathrm{AI}_{\mathrm{L1}}$ on the AMD node is larger, we can infer that higher AIs indicate a stronger likelihood to experience greater parallel speedup. This behavior is consistent with the strong-scaling results of the optimization-based solvers for the Chromium remediation problem in [30] where similar classes of AMD and Intel processors were experimented with.

2.4.2 Demo #2: FEniCS vs Firedrake

Next, we compare the FEniCS and Firedrake implementations of the CG1 element with 16 MPI processes on a single Intel Xeon E5-2680v2 node. The same steady-state diffusion equation is considered, but we now investigate how other multigrid solver packages like HYPRE and ML affect the performance.

The AI's in 2.10 clearly depend on the software implementation, the solver used, and the problem size. The results in this figure suggest that the FEniCS and Firedrake

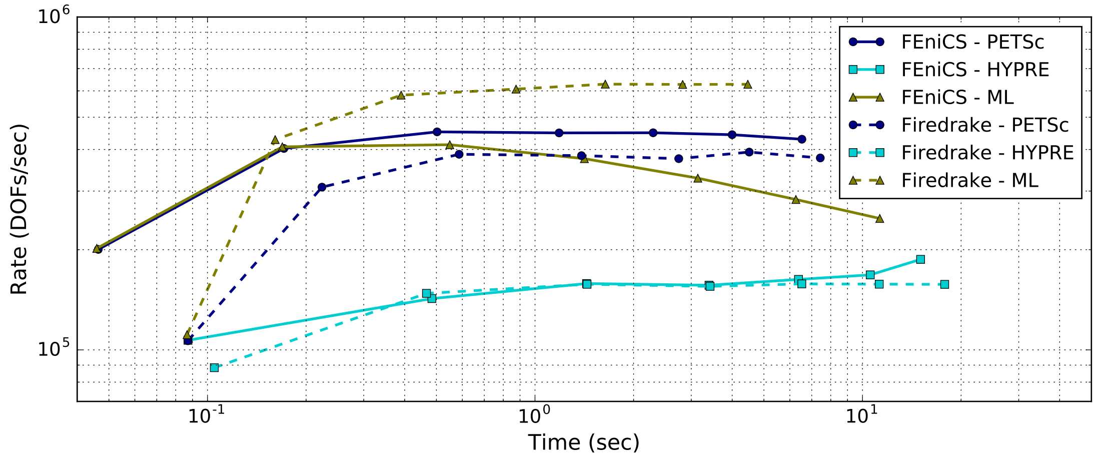


Figure 2.11: Demo #2: Static-scaling for the FEniCS and Firedrake finite element packages with various solver packages on a single Opuntia (Intel Xeon E5-2680v2) node with 16 MPI processes.

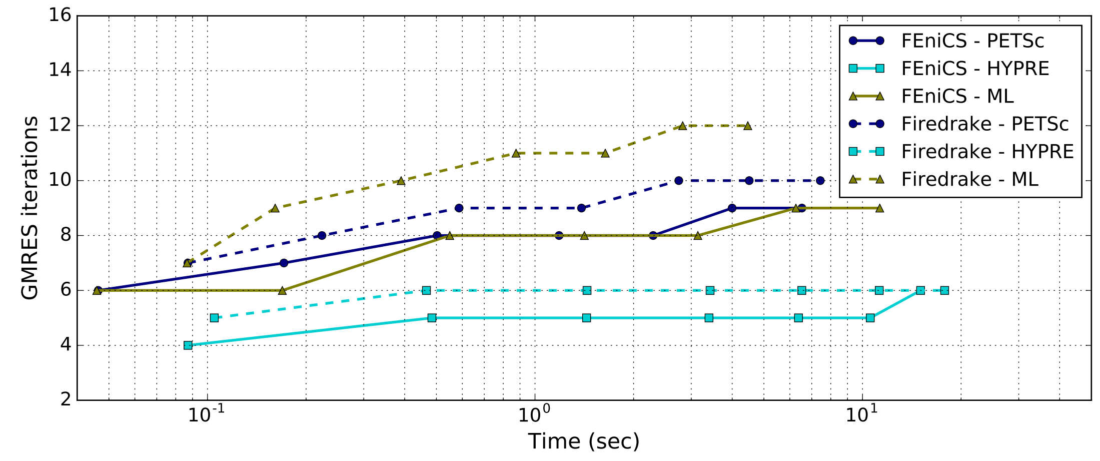


Figure 2.12: Demo #2: Number of GMRES iterations required for the FEnics and Firedrake finite element packages with various solver packages on a single Opuntia (Intel Xeon E5-2680v2) node with 16 MPI processes.

packages have very similar implementations of the PETSc and HYPRE multigrid solvers. However, the $\mathrm{AI}_{\mathrm{L1}}$ for FEniCS's implementation of the ML solver deteriorates rapidly with problem size. Similar behavior is observed in the static-scaling plot of Figure 2.11 where the data points with the highest AI also have the highest rate at which equations are solved. Unlike the previous demonstration where different hardware implementations were compared, the AI and rate metrics are strongly

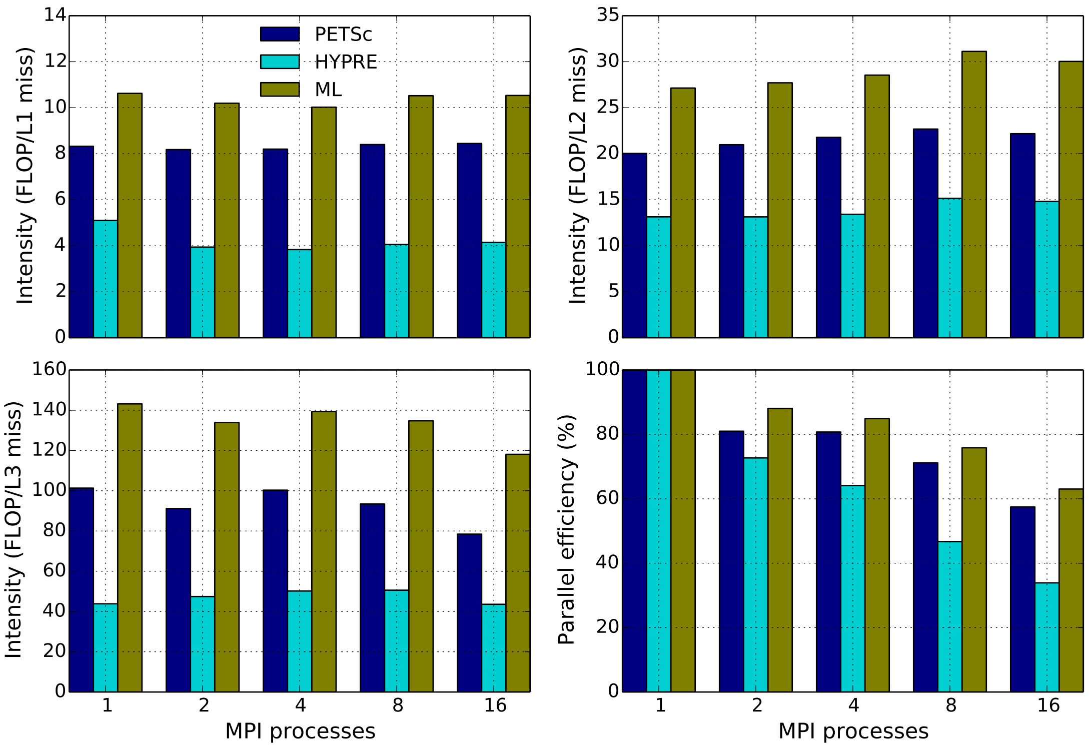


Figure 2.13: Demo #2: Correlation between the L1/L2/L3 arithmetic intensities and strong-scaling efficiency on a single Opuntia (Intel Xeon E5-2680v2) node for up to 16 MPI processes when h-size = 1/140.

correlated to each other, and it is clear that FEniCS's current implementation of ML has some issues since the tailing off towards the right occurs before either the PETSc or HYPRE lines do.

With these two graphs in mind, one may wonder why the tailing off occurs. Does it occur due to suboptimal algorithmic convergence (i.e., iteration count increases with problem size), or do the data structures needed for important solver steps begin to drop out of cache? A scalable algorithm suggests that the number of solver iterations should not increase by much when the problem size increases, so if the GMRES iteration count increases significant, it is possible that the rate metric will decrease. 2.12 denotes the number of GMRES iterations needed for every finite element package and solver, and it can be seen that the iteration counts do not increase by much. These plots must be interpreted carefully because although

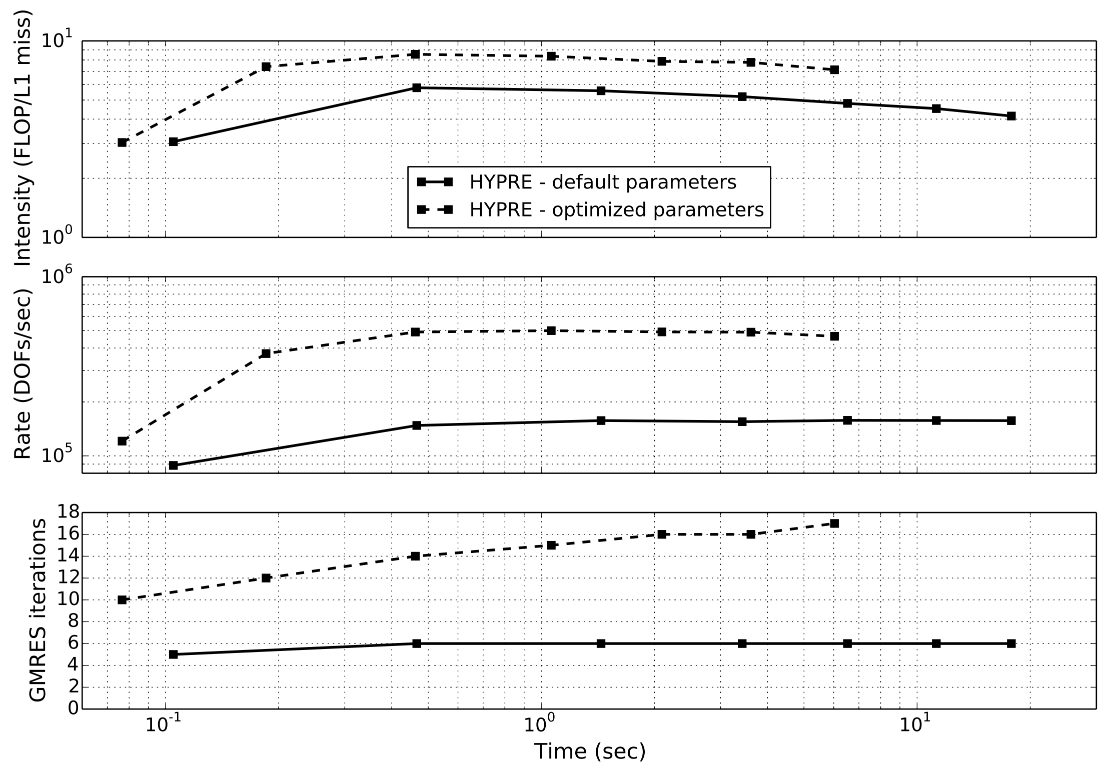


Figure 2.14: Demo #2: Comparison between HYPRE's default solver parameters and HYPRE's optimized solver parameters through the Firedrake package on a single Opuntia (Intel Xeon E5-2680v2) node with 16 MPI processes.

the iteration plots may suggest algorithmic scalability, the degradation in AI with respect to problem size suggests that the current software and solver parameters are not efficiently configured to utilize the hardware. As shown in the previous demonstration, the AI is useful for predicting which algorithms will see greater speedups as the number of MPI processes is increased. Figure 2.13 compares the $\mathrm{AI}_{\mathrm{L}1/2/3}$ and parallel performance of Firedrake's three solver implementations. Regardless of which level of cache is used to determine the AI, HYPRE and ML have the lowest and highest AI's, respectively. Moreover, HYPRE and ML have the worst and best parallel speedups, respectively, which again supports the fact that the AI metric is useful for predicting which algorithms may achieve the greatest parallel speedup.

We note that the HYPRE solver has relatively bad performance, suggesting that the out-of-box parameters are unfit for the problem at hand. One of the best ways to

Table 2.3: Demo #3: Degrees-of-freedom with respect to h-refinement. In this study we do not consider h-size = 1/100 for the DG1 or DG2 elements.

h-size	CG1	CG2	DG1	DG2
1/20	9,261	68,921	192,000	480,000
1/40	68,921	531,441	1,536,000	3,840,000
1/60	226,981	1,771,561	5,184,000	12,960,000
1/80	531,441	4,173,281	12,288,000	30,720,000
1/100	1,030,301	8,120,601	-	-

Table 2.4: Demo #3: L_2 error norm with respect to h-refinement for various finite elements provided through the Firedrake package. In this study we do not consider h-size = 1/100 for the DG1 or DG2 elements.

h-size	CG1	CG2	DG1	DG2
1/20	2.96E-02	3.81E-04	1.65E-02	2.16E-04
1/40	7.77E-03	3.79E-05	4.35E-03	2.26E-05
1/60	3.51E-03	1.06E-05	1.97E-03	6.47E-06
1/80	1.99E-03	4.44E-06	1.12E-03	2.72E-06
1/100	1.28E-03	2.25E-06	-	-
slope:	1.95	3.19	1.94	3.16

improve the AI and Rate$_1$ metrics is to simply adjust some of the solver parameters. If, for example, we optimize the parameters by increasing the strong threshold coarsening rate, the performance improves dramatically as we can tell from Figure 2.14. The AI and Rate$_1$ metrics are now competitive with Firedrake's implementation of the PETSc and ML solvers, but it is important to realize that the GMRES iteration counts increased with size. An algorithm that requires fewer iterations yet remains constant when the problem size increase does not necessarily mean it has good performance and scalability. Neither the AI nor rate metrics tail off towards the right, suggesting that the optimized HYPRE solver is scalable despite some minor growth in the GMRES iteration count. As we have discussed in the previous demonstration, answers regarding performance and scalability of various solvers and software will also depend on the hardware.

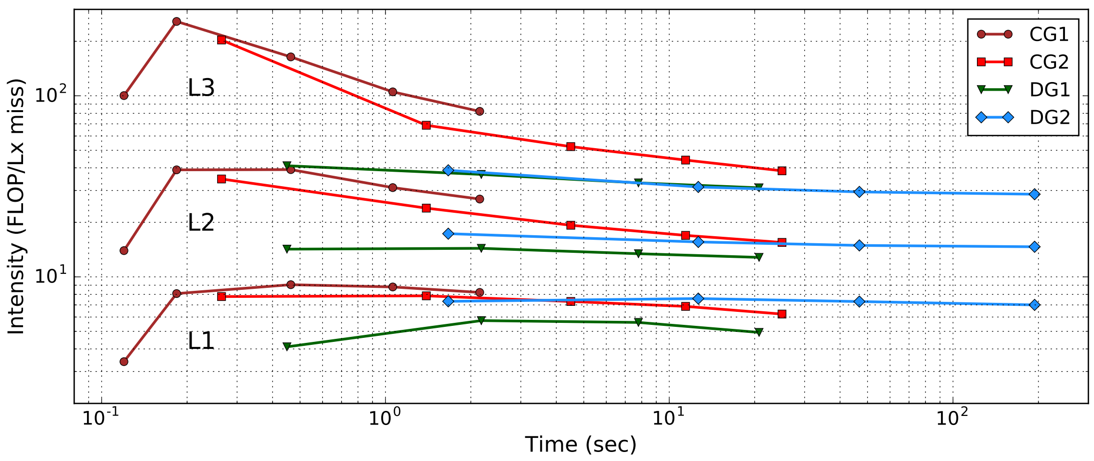


Figure 2.15: Demo #3: L1/L2/L3 arithmetic intensities of Firedrake's various finite element formulations on a single Opuntia (Intel Xeon E5-2680v2) node with 16 MPI processes.

2.4.3 Demo #3: Continuous Galerkin vs Discontinuous Galerkin

So far we have only considered the CG1 finite element. What happens if we employ another discretization such as the Discontinuous Galerkin (DG) method? Moreover, what happens if we increase the polynomial order and employ second order CG (CG2) and second order DG (DG2) elements? Various families of elements and their respective levels of p-refinement will change both the size and numerical accuracy of the numerical solution, so it is desireable to understand both the costs and benefits of these approaches on a particular mesh. Tables 2.3 and 2.4 contain the total degrees-of-freedom and L_2 error norms, respectively, of Firedrake's various finite element discretizations. The CG elements are studied up to h-size $= 1/100$ whereas the DG elements are studied up to h-size $= 1/80$. We again employ 16 MPI processes across a single Intel Xeon E5-2680v2 node, and all finite element discretizations in this demonstration are solved with optimized (i.e., increased strong threshold coarsening) HYPRE parameters.

Figure 2.15 contains the $\text{AI}_{\text{L1/2/3}}$ for the CG1, CG2, DG1, and DG2 elements. What we learn from these results is that increasing the polynomial order for the CG

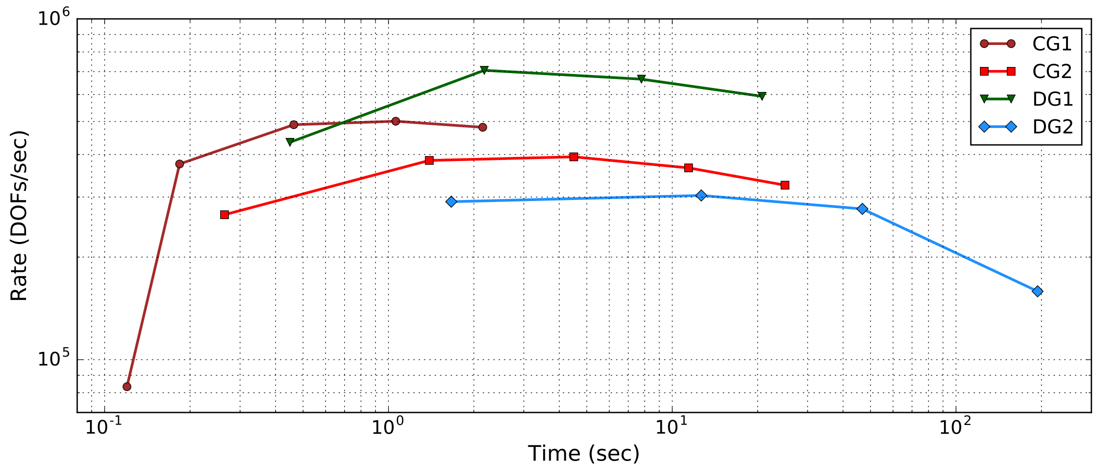


Figure 2.16: Demo #3: Static-scaling for Firedrake's various finite element formulations on a single Opuntia (Intel Xeon E5-2680v2) node with 16 MPI processes.

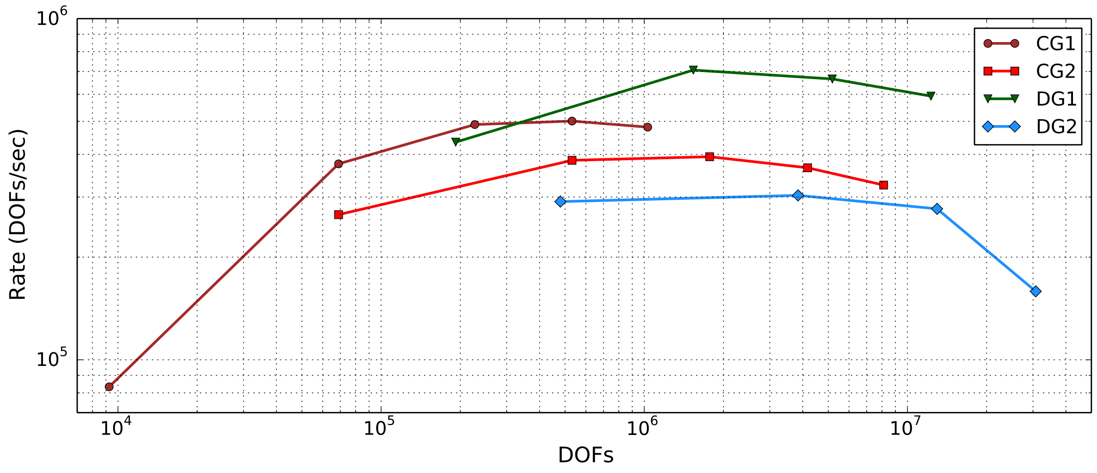


Figure 2.17: Demo #3: Degrees-of-freedom vs degrees-of-freedom solved per second for Firedrake's various finite element formulations on a single Opuntia (Intel Xeon E5-2680v2) node with 16 MPI processes.

elements lowers the AI whereas the AI increases for DG elements. This may not always be the case because different solvers and different hardware architectures may be better tailored to different discretization. Other finite element packages like the FEniCS or DEAL.II projects may have very different results. The Rate$_1$ metric as seen from Figure 2.16 depicts the rate at which each discretization solves its equations. Alternatively, one could also compare the Rate$_1$ metric with respect to the degrees-of-freedom as seen in Figure 2.17. Although DG elements have more degrees-of-freedom

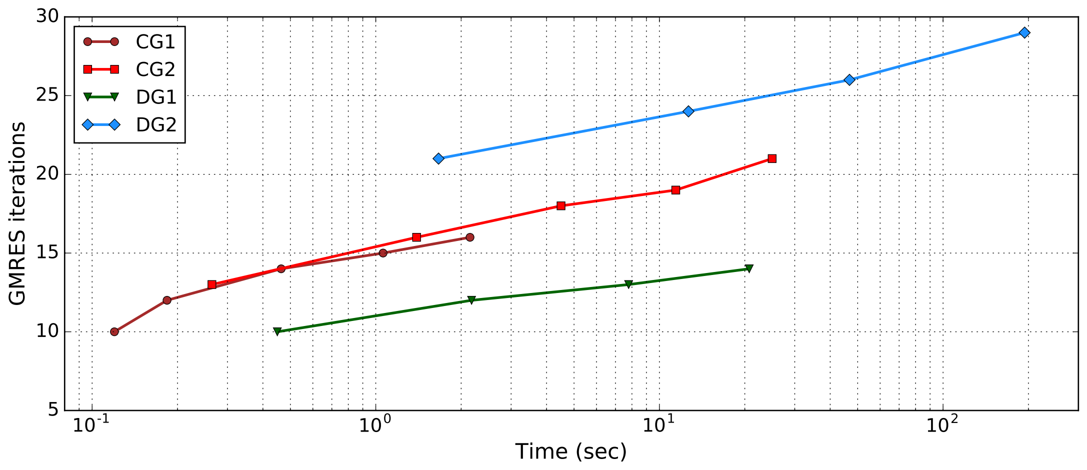


Figure 2.18: Demo #3: Solver iterations needed for Firedrake's various finite element formulations on a single Opuntia (Intel Xeon E5-2680v2) node with 16 MPI processes.

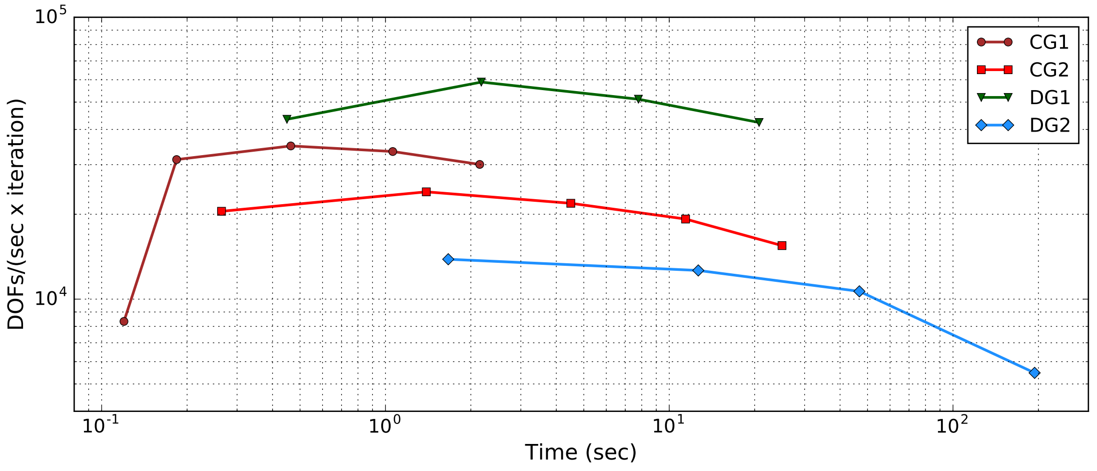


Figure 2.19: Demo #3: Static-scaling per solver iteration for Firedrake's various finite element formulations on a single Opuntia (Intel Xeon E5-2680v2) node with 16 MPI processes.

for a given mesh discretization, it is seen that the DG1 element has the highest Rate$_1$ metric, suggesting that the optimized HYPRE solver parameters are especially suitable for DG1 elements. Unlike the FEniCS and ML combination example from the previous demonstration, the DG2 discretization experiences significant degradation in the static-scaling plot yet maintains relatively consistent AI's. This begs the question of whether the tailing off towards the right is due to memory effects or suboptimal

Table 2.5: Demo #4: L_2 error norm with respect to h-refinement for various values of α in equation (2.4.5) when using the Firedrake implementation of the CG1 element.

h-size	$\alpha = 0$	$\alpha = 1$	$\alpha = 10$	$\alpha = 100$	$\alpha = 1000$
1/20	1.48E-02	3.45E-02	4.83E-02	5.71E-02	5.86E-02
1/40	3.90E-03	9.31E-03	1.46E-02	1.90E-02	1.99E-02
1/60	1.75E-03	4.23E-03	6.84E-03	9.26E-03	9.76E-03
1/80	9.89E-04	2.40E-03	3.94E-03	5.43E-03	5.75E-03
1/100	6.34E-04	1.55E-03	2.55E-03	3.55E-03	3.77E-03
1/120	4.41E-04	1.07E-03	1.78E-03	2.49E-03	2.64E-03
1/140	3.24E-04	7.88E-04	1.31E-03	1.84E-03	1.96E-03
slope:	1.97	1.94	1.86	1.77	1.75

algorithmic convergence.

As previously observed from Figure 2.14, the optimized HYPRE parameters resulted in a slight increase in GMRES iteration count for CG1 elements, and we notice similar trends for the other finite elements in Figure 2.18. If the iteration count increase is significant enough, it could negatively affect static-scaling. To determine whether this solver iteration growth stymied the rate by which equations are solved, we can employ Rate$_2$ (i.e., degrees-of-freedom solved per second per solver iterate) from equation (2.3.6) as shown in Figure 2.19. In this particular demonstration, it makes no difference as we still observe degradation with respect to problem size, hence suggesting that memory bandwidth and cache behavior have an adverse effect on the simulation. Using more compute nodes may certainly ameliorate both the AI and rate metrics for the DG2 element, but it should again be cautioned that comparitive studies on the performance of numerical methods and solvers strongly depend on both the code implementation as well as the nature of the computing platform.

2.4.4 Demo #4: Material properties

So far all three of our demonstrations have been conducted in a homogeneous domain. However, many scientific problems are often hetereogeneous in nature, which

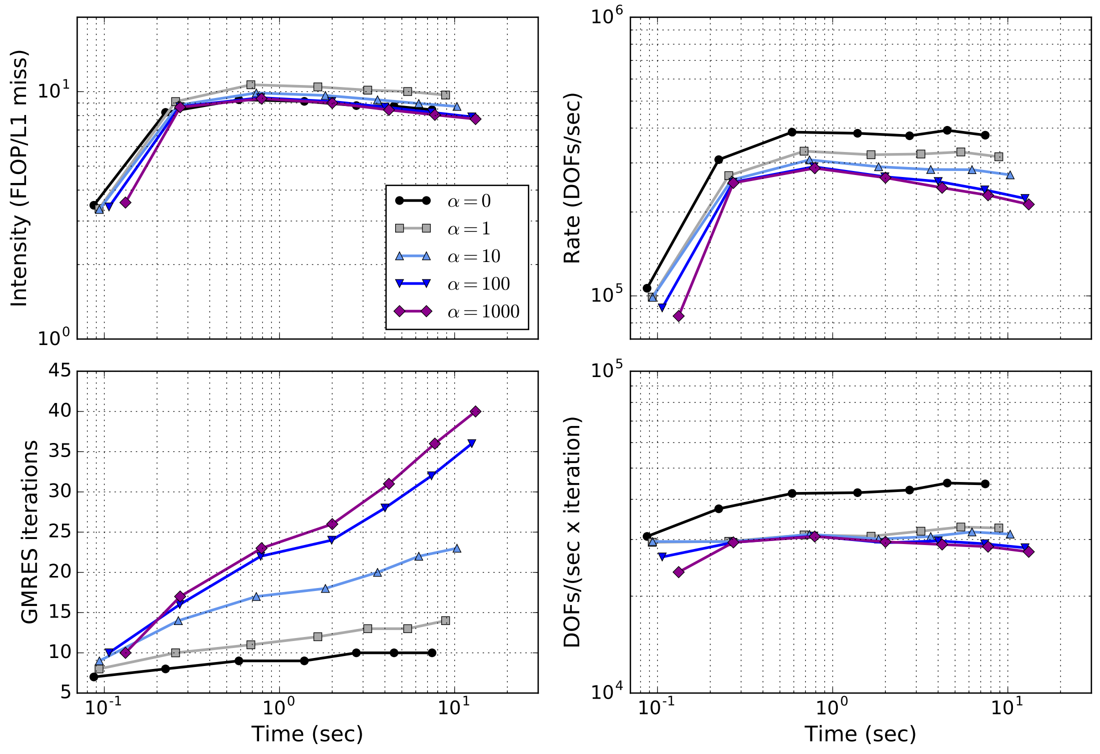


Figure 2.20: Demo #4: Performance spectrum for various values of α in equation (2.4.5) on a single Opuntia (Intel Xeon E5-2680v2) node with 16 MPI processes.

may complicate the physics of the governing equations and may become more expensive to solve numerically. In [31], it was shown that solving heterogeneous problems like chaotic flow resulted in suboptimal algorithmic convergence (i.e., the iteration counts grew with h-refinement), so our goal is to demonstrate how physical properties such a heterogeneity and anisotropy may skew how we interpret the performance. Let us now assume that we have a heterogeneous and anisotropic diffusivity tensor that can be expressed as follows

$$\mathbf{D}(\mathbf{x}) = \begin{pmatrix} \alpha(y^2 + z^2) + 1 & -\alpha xy & -\alpha xz \\ -\alpha xy & \alpha(x^2 + z^2) + 1 & -\alpha yz \\ -\alpha xz & -\alpha yz & \alpha(x^2 + y^2) + 1 \end{pmatrix}, \tag{2.4.5}$$

where $\alpha \geq 0$ is a user defined constant that controls the level of heterogeneity and anisotropy present in the computational domain. By employing the same analytical solution as equation (2.4.3), the various values of α give rise to new forcing functions. The L_2 error norms with respect to α using Firedrake's CG1 elements are shown in Table 2.5. Again a single Intel Xeon E5-2680v2 compute node with 16 MPI processes is used for this study, and PETSc's multigrid solver is used to solve these problems.

Figure 2.20 depicts the AI, Rate$_1$, solver iterations, and Rate$_2$ metrics. The AI is not affected by α which suggests that there are no hardware or software implementation issues, only that the Rate$_1$ metric tails off as α is increased. We see that while the iteration growth is significant, the Rate$_2$ metric is still flat for this heterogeneous and anisotropic steady-state diffusion problem. Thus, the primary reason that the data points in the static-scaling plots decrease with problem size has little to do with memory contention.

2.5 CASE STUDY PART 1: SINGLE NODE

The previous section, which focused entirely on the steady-state diffusion equation, covered the basic ways one can utilize the proposed performance spectrum model to help justify, interpret, or diagnose the computational performance of any algorithm, numerical method, or solver for a particular compute node. In these next two sections, we demonstrate that this performance spectrum model is also useful for more complicated and nonlinear PDEs. We consider PETSc's toy hydrostatic ice sheet flow example, based on the work of [26], with geometric multigrid and apply the performance spectrum to give us a better understanding of how certain HPC platforms scale.

2.5.1 Hydrostatic ice sheet flow equations

Consider a $[0,10]\text{km} \times [0,10]\text{km} \times [0,1]\text{km}$ computational ice domain $\Omega \subset \mathbb{R}^3$ lying between a Lipschitz continuous bed $b(x,y)$ and surface $s(x,y)$. The hydrostatic equations are obtained from the non-Newtonian Stokes equations where the horizontal $x-$ and $y-$ derivatives of velocity in the vertical z-direction are small and negligible. Denoting the horizontal component of the velocity field by $\boldsymbol{u} = (u,v)$ where u and v are parallel to the $x-$ and $y-$ axes respectively, the governing equations for hydrostatic ice sheet flow is given by

$$-\eta\left(\frac{\partial}{\partial x}\left(4\frac{\partial u}{\partial x}+2\frac{\partial v}{\partial y}\right)+\frac{\partial}{\partial y}\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)+\frac{\partial^2 u}{\partial z^2}\right)+\rho g\frac{\partial s}{\partial x}=0 \quad \text{and} \tag{2.5.1a}$$

$$-\eta\left(\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)+\frac{\partial}{\partial y}\left(2\frac{\partial u}{\partial x}+4\frac{\partial v}{\partial y}\right)+\frac{\partial^2 y}{\partial z^2}\right)+\rho g\frac{\partial s}{\partial y}=0, \tag{2.5.1b}$$

where η is the nonlinear effective viscosity expressed by

$$\eta(\gamma)=\frac{B}{2}\left(\frac{\epsilon^2}{2}+\gamma\right)^{\frac{1-n}{2n}}, \tag{2.5.2}$$

where ice sheet models typically take $n = 3$. The hardness parameter is denoted by B, the regularizing strain rate is defined by ϵ, and the second invariant γ is expressed by

$$\gamma=\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}+\frac{\partial u}{\partial x}\frac{\partial v}{\partial y}+\frac{1}{4}\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)^2+\frac{1}{4}\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 v}{\partial x^2}\right). \tag{2.5.3}$$

More information on the theoretical derivation of the above equations can be found in [148, 149]. Equation (2.5.1) is subject to natural boundary conditions at the free surface and either no-slip or power-law slip conditions with friction parameter

$$\beta^2(\gamma_b)=\beta_0^2\left(\frac{\epsilon_b^2}{2}+\gamma_b\right)^{\frac{m-1}{2}}, \tag{2.5.4}$$

where $\gamma_b = \frac{1}{2}\left(u^2 + v^2\right)$, ϵ_b is regularizing velocity, β_0^2 is a “low-speed" reference friction, and $m \in (0, 1]$ is the slip exponent.

2.5.2 Problem setup

The hydrostatic ice sheet flow equation is discretized using hexahedron Q1 finite elements on a structured grid and Figure 2.21 contains the corresponding solution. Details concerning the theoretical derivation of the variational formulation as well as the parameters used for the boundary value problem can be found in [26]. Since this example problem is written entirely with PETSc routines and function calls, the [FLOPs] metric in equation (2.3.4) is determined using PETSc's manual FLOP counts instead of hardware FLOP counters. This is particularly useful if a thorough comparative study on PETSc's eclectic suite of linear algebra solvers for a particular PDE were to be conducted.

For this problem, we begin with an initial coarse grid size and successively refine the grid N times until we get the desired problem size and numerical accuracy. The “fine grids" produced from this element-wise refinement are solved using the geometric multigrid technique, whereas the initial coarse grid is solved using algebraic multigrid. The assembled Jacobian employs block AIJ format (better known as the compressed sparse row format), where the horizontal velocity components are grouped per grid node. Since ice-flow is tightly coupled in the vertical direction, parallel domain decomposition is specially set up so that grid points in the vertical direction are never distributed and are always contiguous in memory. The initial grid size must be chosen carefully because the mesh partition happens at the coarsest level and may cause load balancing issues if the initial grid is not large enough.

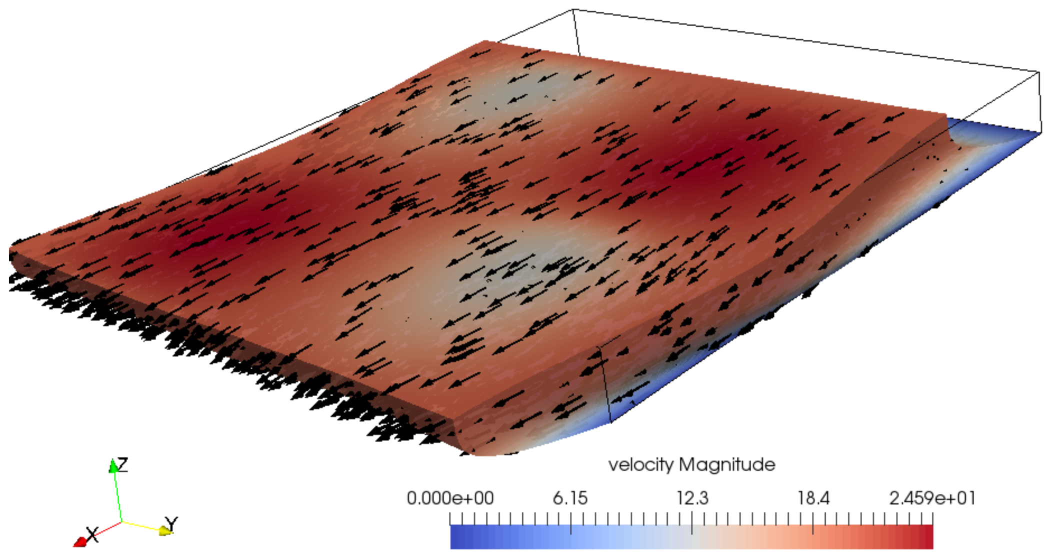


Figure 2.21: Numerical solution of the velocity vector field for the hydrostatic ice sheet flow example.

Table 2.6: Hydrostatic ice sheet flow for a single node: Mesh information and number of solver iterations needed for an initial 40×40×5 coarse grid. KSP and SNES iteration counts may vary depending on the number of MPI processes used.

Levels of refinement	Degrees-of-freedom	SNES iterations	KSP iterations
0	16,000	7	39
1	115,200	8	45
2	870,400	8	44
3	6,758,400	8	44

2.5.3 Results

First, we provide an initial 40×40×5 coarse grid and successively refine this grid up to 3 times. All five processors from Table 2.1 are studied, and the KNL processor is configured to use MCDRAM in flat mode. Each node has a different number of available cores so in order to maintain relatively consistent mesh partitioning, the Ivybridge and KNL processors will only utilize 16 and 64 cores, respectively. Table 2.6 presents the problem size as well as the number of total SNES and KSP iterations needed for each level of refinement. Figure 2.22 depicts the $\mathrm{AI}_{\mathrm{L1}}$ metrics with respect to the overall time-to-solution. Each data point has the same coarse grid

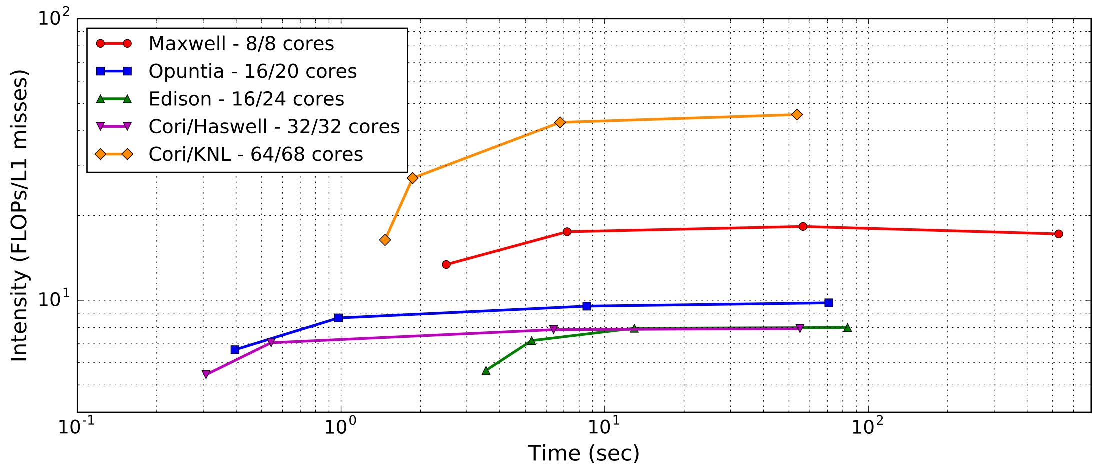


Figure 2.22: Hydrostatic ice sheet flow single node: L1 arithmetic intensity, based on PETSc's manual FLOP counts and L1 cache misses. Note that the Ivybridge (i.e., Opuntia and Edison) and KNL nodes are only partially saturated.

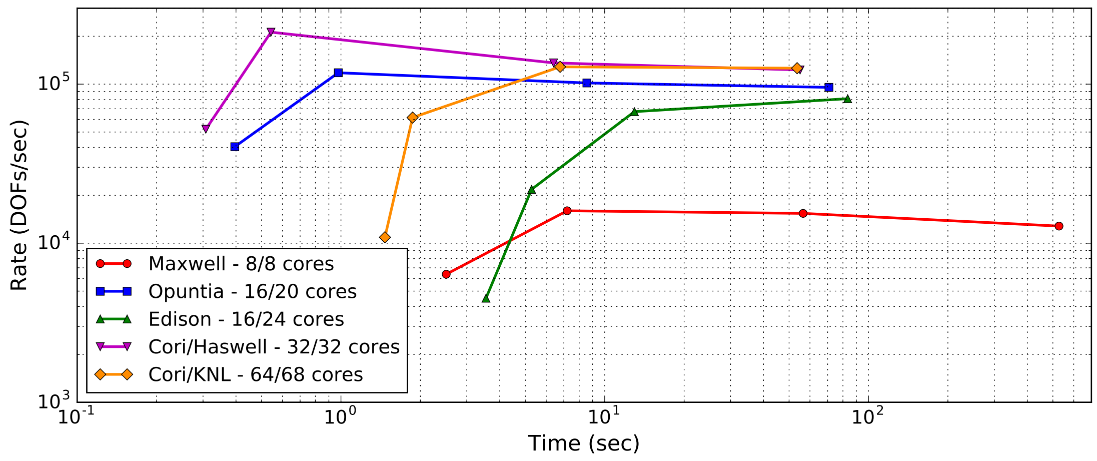


Figure 2.23: Hydrostatic ice sheet flow single node: Static-scaling. Note that the Ivybridge (i.e., Opuntia and Edison) and KNL nodes are only partially saturated.

size but has different levels of grid refinement ranging from 0 to 3. The "Ivybridge" and "Haswell" processors have similar AIs and are significantly smaller than their KNL and AMD counterparts. It should be noted that GNU compilers were used to compile the problem on the AMD Opteron 2354 and Intel Xeon E5-2680v2 processors whereas the other three processors used Cray compilers, which could explain why the AIs between the two Ivybridge processors are slightly different. As with the hardware

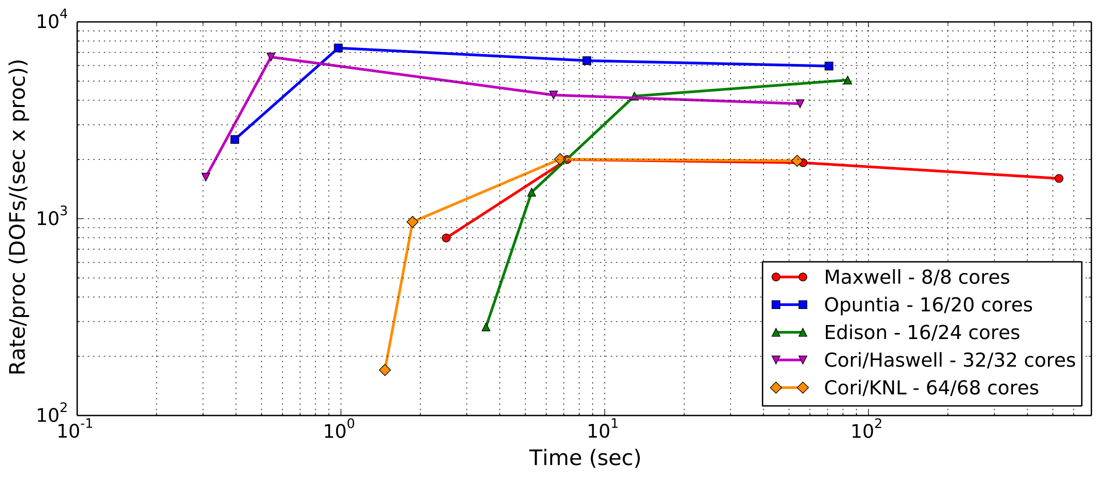


Figure 2.24: Hydrostatic ice sheet flow single node: Static-scaling per MPI processes. Note that the Ivybridge (i.e., Opuntia and Edison) and KNL nodes are only partially saturated.

counter examples in the last section, the AIs are initially small for the coarser problems but eventually stabilize if a problem is sufficiently large. It is interesting to note that the AMD processor is consistently flat for all data points, suggesting that the smaller problem sizes selected for this example have already approached the AMD processor's achievable peak performance of the computational/memory resource. On the other hand, the KNL's wider vector instruction sets and caches for smaller problems are not fully utilized, resulting in low AI's.

The static-scaling plot on each of these compute nodes is shown in Figure 2.23, and Figure 2.24 depicts static-scaling based on Rate$_3$ from (2.3.7). Unsurprisingly, the AMD processor is outperformed by all of the Intel processors. Both of Cori's Intel processors have the best performance out of all the systems studied, but we again notice that the KNL processor has poor metrics when the grid is small. Furthermore, KNL's performance per core is considerably lower as seen from Figure 2.24. There are many reasons why we noticed such dramatic behavior. First, as we already noted from the AI results, the problem has to be sufficiently large in order to fully utilize the KNL vector instructions. Second, we used 64 of the 68 available cores on the KNL node, which is at least double the amount of cores that the other systems have.

Table 2.7: Hydrostatic ice sheet flow strong-scaling for an initial 64×64×6 coarse grid.

Nodes	Edison			Cori/Haswell			Cori/KNL		
	Cores	Time (s)	Eff. (%)	Cores	Time (s)	Eff. (%)	Cores	Time (s)	Eff. (%)
1	16	300	-	32	227	-	64	193	-
2	32	150	100	64	108	105	128	92.4	104
4	64	72.0	104	128	57.3	99.0	256	47.3	102
8	128	37.9	98.9	256	28.7	98.9	512	25.2	95.7
16	256	18.9	99.2	512	13.9	100	1024	15.1	79.9
32	512	9.65	97.2	1024	8.11	87.5	2048	10.6	56.9
64	1024	6.75	69.4	2048	4.62	76.8	4096	9.27	32.5

Table 2.8: Hydrostatic ice sheet flow for multiple nodes: Mesh information and number of solver iterations needed for an initial 128×128×12 coarse grid. KSP and SNES iteration counts may vary depending on the number of MPI processes used.

Levels of refinement	Degrees-of-freedom	SNES iterations	KSP iterations
1	3,014,656	8	85
2	23,592,960	8	85
3	186,646,528	8	85
4	1,484,783,616	8	85
5	11,844,714,496	8	85

The degrees-of-freedom per MPI is significantly smaller so it is possible interprocess communication time affects the scaling results.

2.6 CASE STUDY PART 2: MULTIPLE NODES

The results from every example in this chapter thus far behoove us to now investigate what happens when more than one compute node is needed to solve a PDE. Figures 2.9 and 2.11 from the previous section indicate that the $\mathrm{AI}_{\mathrm{Lx}}$ metrics can be used to predict the strong-scaling potential on a single node. Our goal is now to investiate if the correlation holds true even across multiple nodes. To ensure that the problem is sufficiently large to distribute to several nodes, we consider an initial 64×64×6 coarse grid with three levels of refinement (21,495,808 degrees-of-freedom). The number of KSP and SNES iterations needed to solve the problem are 62 and 8, respectively.

Table 2.7 contains the strong-scaling results for the Cori and Edison systems.

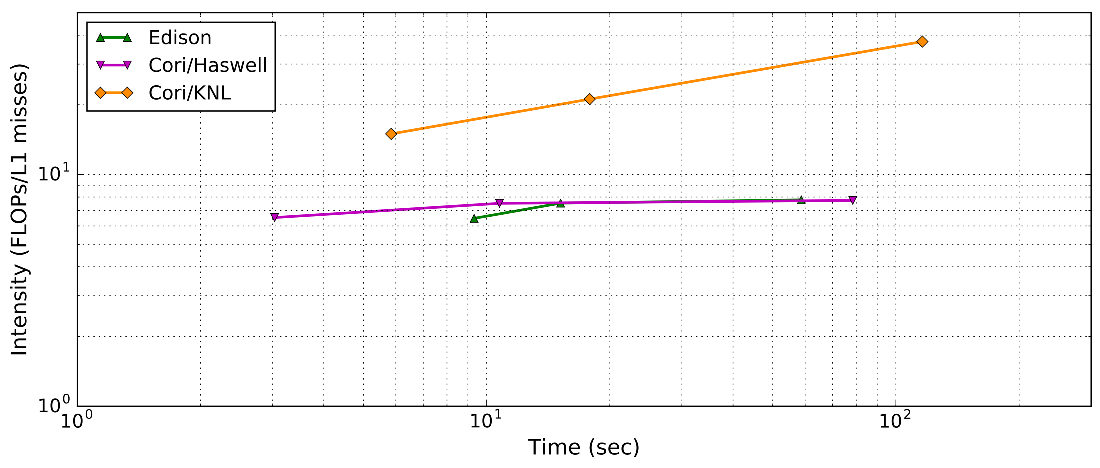


Figure 2.25: Hydrostatic ice sheet flow multiple nodes: AI_{L1} when the systems all employ 1024 cores (64 Edison nodes, 32 Haswell nodes, and 16 KNL nodes). Grid sizes ranging from 3 million to 186 million degrees-of-freedom are considered.

All three systems demonstrate near perfect strong-scaling performance until 1024 cores are used (roughly 20k degrees-of-freedom per core). However, it is difficult to make performance comparisons because different systems employ different numbers of MPI processes per node which affect communication to computation ratios as well as required data bandwidth between nodes. The only concrete conclusion that can be made is that the KNL system takes the least amount of wall-clock time on a single compute node but gets outperformed when the problem size per node reduces. Figures 2.22 and 2.23 suggest that when the problem size on a KNL node is sufficiently small, parallel performance would degrade drastically, which is exactly what the results of Table 2.7 portray.

2.6.1 Example #1: 1024 MPI processes

In this section, we consider what happens when we employ the same MPI concurrency. This second example aims to model the performance when the same hydrostatic ice sheet flow problem is solved on Cori and Edison systems each utilizing 1024 MPI processes. We set this problem up by allocating 64 Edison/Ivybridge nodes, 32 Cori/Haswell nodes, and 16 Cori/KNL nodes. An even larger initial 128×128×12

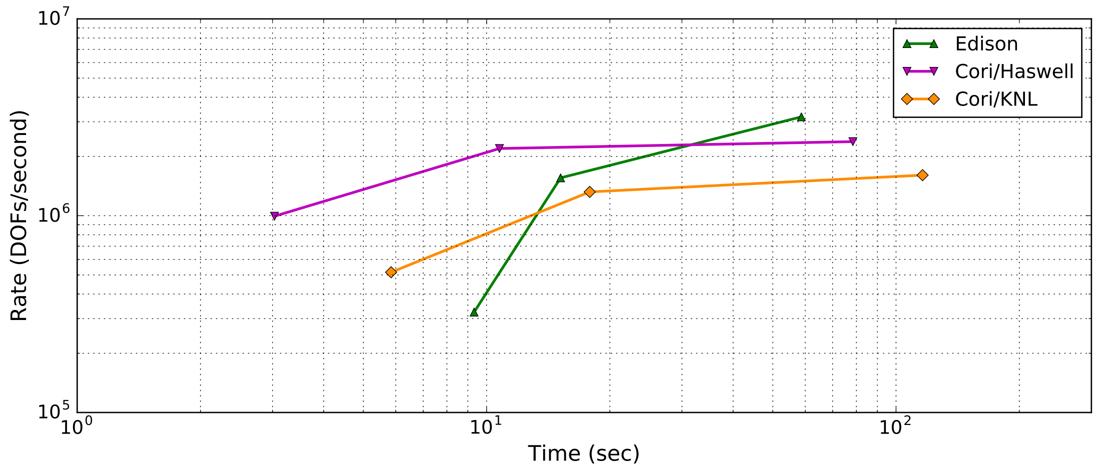


Figure 2.26: Hydrostatic ice sheet flow multiple nodes: Static-scaling when the systems all employ 1024 MPI processes (64 Edison nodes, 32 Haswell nodes, and 16 KNL nodes). Three levels of refinement are considered.

coarse grid is selected, and we refine the problem 1–3 times. Table 2.8 presents the problem size as well as the number of nonlinear and linear solver iterations needed for every level of refinement. Figures 2.25 and 2.26 contain the intensity and rate metrics, respectively. The AI data points are either relatively flat or do not experience drastic changes upon mesh refinement. The static-scaling plot tells us that the Edison/Ivybridge system has the best performance as the problem gets larger. This behavior may seem to contradict the findings of the static-scaling plot in Figure 2.23, but it is important to realize that this PETSc application is limited by the memory-bandwidth and not the TPP for the FLOP rate. The HPC system with Ivybridge processors has the best performance simply because it employs more compute nodes thus more available memory.

2.6.2 Example #2: 256 compute nodes

The previous example is an elegant demonstration of why comparing HPC machines based on equal MPI concurrency can produce misleading performance metrics, especially for computational frameworks that are limited by the memory-bandwidth. What happens if every system employs the same number of compute nodes? In this

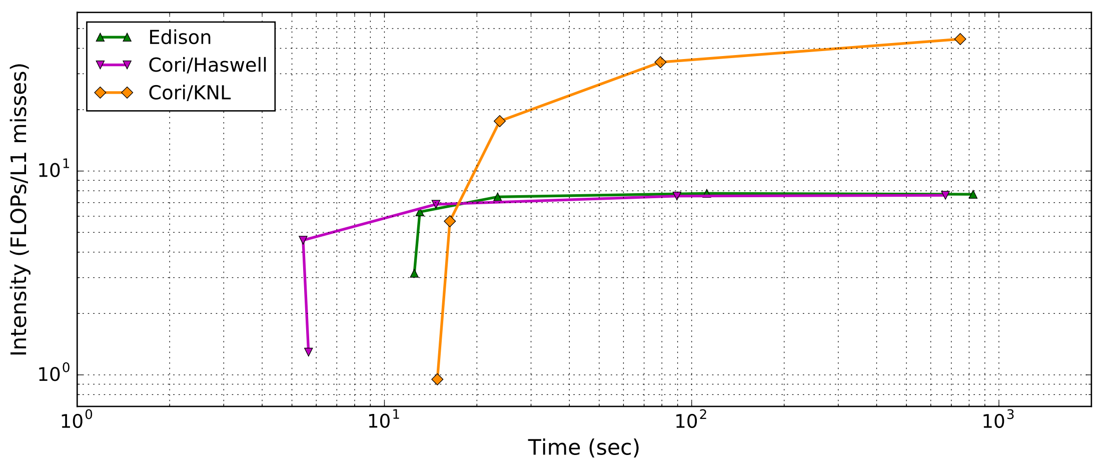


Figure 2.27: Hydrostatic ice sheet flow multiple nodes: AI_{L1} when the systems all employ 256 nodes (4096, 8192, and 16384 MPI processes for Edison, Haswell, and KNL respectively). Five levels of refinement are considered.

third example, the Cori and Edison systems shall allocate 256 compute nodes each. Thus, Edison/Ivybridge will use 4096 MPI processes (16 out of 24 cores per node), Cori/Haswell will use 8192 MPI processes (32 out of 32 cores per node), and Cori/KNL will use 16384 processes (64 out of 68 cores per node). We use the same initial coarse grid as in the previous example but now refine the problem 1–5 times. The AI in Figure 2.27 again indicate relative consistency for finer problems, and we again observe that the AI metric will drop significantly if a problem is not large enough. This trend corroborates the notion that the AI dropping for small problems happens regardless of whether a single node or multiple nodes are used. The static-scaling plot shown in Figure 2.28 demonstrates that Edison's Ivybridge processor does not beat out Cori's Haswell processor. What's particularly interesting is that the performance for Cori's KNL processor drastically varies with problem size. KNL cannot beat out Edison for small problems, but KNL will beat both Edison and Haswell when a problem is neither too small nor too large.

The performance spectrum model is useful for understanding performance characteristics across a wide variety of hardware architectures. Although the STREAM

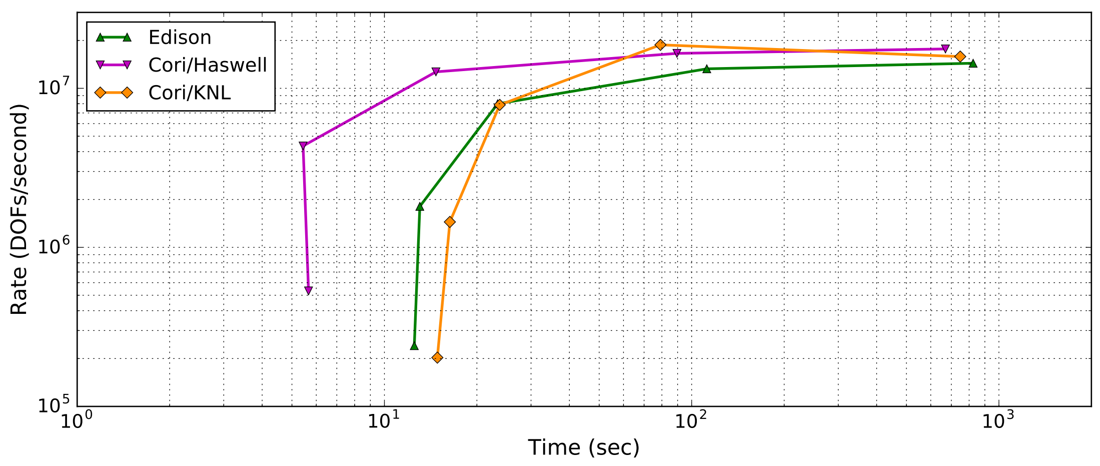


Figure 2.28: Hydrostatic ice sheet flow multiple nodes: Static-scaling when the systems all employ 256 nodes (4096, 8192, and 16384 MPI processes for Edison, Haswell, and KNL respectively). Five levels of refinement are considered.

Triad measurements from Table 2.1 suggest that KNL should greatly outperform Ivybridge and Haswell for memory-bandwidth dominated applications, the performance spectrum indicates that current and practical implementations of scientific software like PETSc v3.7.4 on KNL may be slow if the problem is dominated by main memory bandwidth. Different platforms require different implementation methodologies in order to maximize performance, so optimizing computational frameworks to fully leverage the power of the KNL processor is still an open research problem. Nonetheless, the performance spectrum model is useful for testing various implementations of PDE solvers and can be utilized to understand hardware architectures trends and algorithms of the future.

2.7 CONCLUDING REMARKS

In this chapter, we have proposed a performance model, referred to as the *performance spectrum*, designed to simultaneously model both the hardware/architectural and algorithmic efficiencies of a variety of parallel PDE solvers. The techniques needed to approximate such efficiency metrics are 1) the arithmetic intensity documenting the ratio of flops over data cache misses, and 2) static-scaling, which scales up the problem while fixing the concurrency. This model enabled us to visualize and enrich our current understanding of performance issues related to hardware limitations, software/solver implementation, and numerical discretization of some popular and state-of-the-art finite element simulation packages and solvers. Moreover, it has been shown that this spectrum is also useful for understanding performance and scalability of complex solvers and PDEs for nonlinear problems like hydrostatic ice sheet flow in a large-scale environment. Computational scientists have designed and are still designing software and algorithms needed to answer many of today's pressing scientific problems, so not only do we need to solve these problems accurately but also to solve them fast. In order to understand how fast these solvers and software are, particularly ones that are either black-box or designed by others, we need a performance model, such as the proposed performance spectrum, to help answer any questions regarding computational performance.

Chapter 3. Large-scale optimization-based non-negative computational framework for diffusion equations: Parallel implementation and performance studies

3.1 INTRODUCTION

The modeling of flow and transport in subsurface is vital for energy, climate and environmental applications. Examples include CO_2 migration in carbon-dioxide sequestration, enhanced geothermal systems, oil and gas production, radio-nuclide transport in a nuclear waste repository, groundwater contamination, and thermo-hydrology in the Arctic permafrost due to the recent climate change [79, 101, 67, 81]. Several numerical codes (e.g., FEHM [167], TOUGH [136], PFLOTRAN [100]) have been developed to model flow and transport in subsurface at reservoir-scale. These codes typically solve unsteady Darcy equations for flow and advection-diffusion equation for transport. The predictive capability of a numerical simulator depends on the robustness of the underlying numerical methods. A necessary and essential requirement is to satisfy important mathematical principles and physical constraints. One such property in transport and reactive-transport problems is that the concentration of a chemical species cannot be negative. Mathematically, this translates to the satisfaction of the discrete maximum principle (DMP) for diffusion-type equations. Subsurface flow and transport applications typically encounter geological media that are highly heterogeneous and anisotropic in nature, and it is well-known that the classical finite element (or finite volume and finite difference, for that matter) formulations do not produce non-negative solutions on arbitrary meshes for such porous media.

3.1.1 Prior works on non-negative formulations

Before we discuss the prior works, it is important to discuss the reason behind the negative solutions under the conventional methods. The precise mathematical answer is that the conventional methods do not converge *uniformly* to the exact solution, which will be the case under strong anisotropic dispersion. In fact, the paper by [37] also provides the same reason in their study on discrete maximum principles of the Poisson's equation. The uniform convergence issues also manifest in the case of steep gradients and one such case is the famous Gibbs phenomenon [58, 59]. It is difficult to explain intuitively why the Gibbs phenomenon occurs or why some sequence of functions do not converge uniformly. It is just the nature of the approximation of functions.

The prior non-negative formulations can be broadly classified into the following five categories:

1. *Reporting the violations*: In [134], several cases of violations of the maximum principle and the non-negative constraint have been showcased for different anisotropic diffusivity tensors. This chapter also demonstrates that *h*- and *p*-refinements do not eliminate these violations. The adverse effects due to violations of the non-negative constraint for non-linear ecological models and chemically reacting flows have been illustrated in [122]. These mentioned papers and the references therein have clearly documented that the violations of the non-negative constraint need not be small, which is especially true in the case of anisotropic diffusion. However, neither of these papers have provided any fix to overcome these violations.

2. *Mesh restrictions*: The first work on maximum principles under the finite element method can be traced back to the seminal paper by [37]. This paper considered *isotropic* diffusion, and has shown that an acute-angled triangular mesh

(which is a restriction on the mesh) will satisfy the maximum principle under the finite element method. Anisotropic diffusion equations have been addressed in [78], wherein they developed an algorithm to generate metric-based meshes to satisfy the maximum principle for such equations. [121] have addressed various versions of maximum principles for diffusion and advection-diffusion equations and studied the performance of metric-based meshes for these equations. This paper highlighted the main deficiency of metric-based meshes, which is the need to alter the mesh for different diffusivity tensors. A comprehensive list and discussion of other prior works related to enforcing mesh restrictions to meet the maximum principle and the non-negative constraint can also be found in [121].

3. *Developing or altering formulations in the continuum setting*: Two works that fall under this category are [69, 133], both of which addressed transient transport problems. [69] utilized a stabilized method that is available for Helmholtz-type equations to construct a stabilized formulation for transient *isotropic* diffusion equations to meet the maximum principle. This approach, as presented in [69], is applicable to one-dimensional setting. [133] meets maximum principles for transient transport equations by employing two techniques. They rewrote transient transport equations, which are parabolic in nature, into modified Maxwell-Cattaneo equations, which are hyperbolic in nature, and employed the *space-time* Discontinuous Galerkin approach.

4. *Non-finite element approaches*: A finite volume-based approach, to enforce the non-negative constraint, as proposed in [135], involves a non-linear iterative procedure to select appropriate collocation points for cell concentrations. This technique has been refined by several others including [104] and [152]. Other similar approaches include the mimetic finite difference method [40], which ensures monotonicity and positivity. Since neither the finite difference nor finite

volume methods are based on weak formulations, these mentioned works cannot be trivially extended to the finite element method.

5. *Optimization-based techniques at the discrete level*: Several studies over the years [132, 127, 130, 131] have focused on the development of optimization-based methodologies that enforce the maximum principle and the non-negative constraint for diffusion problems. An optimization-based methodology based on the work of the aforementioned studies has been applied to enforce maximum principles advection-diffusion equations [120]. By reformulating the advection-diffusion problem as a mixed finite element formulation under the least-squares formalism, one introduces flux variables into the problem. The discrete formulation is also symmetric and positive-definite, so one can easily apply both bounded constraints and equality constraints to ensure non-negative solutions and local mass conservation respectively. It should be noted that one may also employ normal equations or the least-squares approach to ensure that the minimization problem for non-symmetric problems is convex [43, 28, 133]. All of these studies have employed quadratic programming (QP) techniques to enforce the maximum principle on 2D academic problems, but the problems studied are small-scale and do not require state-of-the-art Krylov subspace (KSP) iterative solvers and preconditioners. Moreover, it is difficult to find solvers for least-squares or penalty-type problems due to the condition numbers of large systems of equations [137].

3.1.2 Large-scale computing

Although the aforementioned studies have successfully enforced the discrete maximum principles and the non-negative constraint, they did not address how such methods can be used for realistic large-scale subsurface problems that have millions of grid nodes. Furthermore, complex coupling between different physical processes as well as the presence of multiple species amplify the degrees-of-freedom (i.e., the number of unknowns). The aim of this chapter is to develop a parallel computational framework that solves anisotropic diffusion equations on general meshes, ensures non-negative solutions, and can be employed to solve large-scale realistic problems.

Large-scale problems can be tackled by using recent advancements in high-performance computing (HPC) methods and toolkits that can be used on the state-of-the-art supercomputing architecture. One such toolkit is PETSc [12], which provides data structures and subroutines for setting up structured and unstructured grids, communication routines for using clusters and massively parallel processes (MPP), linear and non-linear solvers, and parallel I/O. These high-level data structures and subroutines help in faster development of parallel application codes and minimize the need to program low-level message passing and partition data structures, so that the domain scientists can focus mostly on the application. To this end, we develop a parallel framework satisfying the DMP by leveraging the existing capabilities within PETSc. This property is obtained for anisotropic diffusion by using lower-order finite elements and the optimization-based approach in [132, 105, 127]. The TAO toolkit [123], which is built on top of PETSc, has a wide suite of commonly-used optimization solvers. The robustness of the proposed framework will be demonstrated by solving realistic large-scale problems.

It needs to be mentioned that there are other possible ways of solving the optimization problem at hand in a large-scale setting, and a brief discussion is warranted on this aspect. Herein, we pose the problem as a quadratic programming problem and employ tools that are primarily developed to handle such optimization problems. However, one can also pose the problem as a variational inequality problem and in particular as an obstacle problem [84]. A thorough discussion on the mathematical aspects of variational inequalities can be found in [84, 50]. The state-of-the-art computational strategies for solving large-scale variational inequalities can be found in [15, 160], and a discussion on the associated linear solvers and preconditioners can be found in [62]. A systematic study of the problem at hand using variational inequalities will be addressed in the next chapter.

The rest of this chapter is organized as follows. In Section 3.2, we present the governing equations and the classical single-field Galerkin finite element formulation for steady-state and transient diffusion equations. The optimization-based method to ensure non-negative concentrations is also outlined in this section. In Section 3.3, the parallel implementation procedure using PETSc and TAO is presented. We also highlight the relevant data structures used in this study and present a pseudo algorithm describing our parallel framework. In Section 3.4, a performance model loosely based on the roofline model is outlined. This model is used to estimate the efficiency with respect to computing hardware utilization of currently available solvers within PETSc and TAO. In Section 3.5, we first verify our implementation using a 3D benchmark problem from the literature and present a detailed performance study using the proposed model. Then, we study a large-scale three-dimensional realistic problem involving the transport of chromium in the subsurface and document the numerical results of the non-negative methodology with the classical single-field Galerkin formulation. Conclusions are drawn in Section 3.6.

We shall denote all the continuum vectors by lowercase boldface unitalicized letters (e.g., $\mathbf{a}$), and the vectors in the discrete setting are denoted by lowercase boldface italic letters (e.g., $\boldsymbol{a}$). We shall denote all the continuum second-order tensors by boldface uppercase unitalicized letters (e.g., $\mathbf{A}$), and all the finite element matrices are denoted by uppercase boldface italicized letters (e.g., $\boldsymbol{A}$). Other notational conventions are introduced as needed in the chapter.

3.2 GOVERNING EQUATIONS AND ASSOCIATED NON-NEGATIVE NUMERICAL METHODOLOGIES

Let $\Omega \subset \mathbb{R}^{nd}$ be a bounded open domain, where "nd" is the number of spatial dimensions. A spatial point is denoted by $\mathbf{x} \in \overline{\Omega}$. The boundary of the domain is denoted by $\partial\Omega = \overline{\Omega} - \Omega$, which is assumed to be piecewise smooth. The gradient and divergence operators with respect to $\mathbf{x}$ are, respectively, denoted as grad$[\cdot]$ and div$[\cdot]$. As usual, the boundary is divided into two parts: Γ^{D} and Γ^{N}. Γ^{D} is that part of the boundary on which Dirichlet boundary conditions are prescribed, and Γ^{N} is the part of the boundary on which Neumann boundary conditions are prescribed. For mathematical well-posedness, we assume $\Gamma^{\mathrm{D}} \cup \Gamma^{\mathrm{N}} = \partial\Omega$ and $\Gamma^{\mathrm{D}} \cap \Gamma^{\mathrm{N}} = \emptyset$. The unit outward normal to boundary is denoted as $\widehat{\mathbf{n}}(\mathbf{x})$. The diffusivity tensor is denoted by $\mathbf{D}(\mathbf{x})$, which is assumed to be symmetric, bounded above and uniformly elliptic. That is,

$$\mathbf{D}(\mathbf{x}) = \mathbf{D}^{\mathrm{T}}(\mathbf{x}) \quad \forall \mathbf{x} \in \Omega, \tag{3.2.1}$$

and there exists two constants $0 < \xi_1 \leq \xi_2 < +\infty$ such that

$$\xi_1 \mathbf{y}^{\mathrm{T}}\mathbf{y} \leq \mathbf{y}^{\mathrm{T}}\mathbf{D}(\mathbf{x})\mathbf{y} \leq \xi_2 \mathbf{y}^{\mathrm{T}}\mathbf{y} \quad \forall \mathbf{x} \in \Omega \text{ and } \forall \mathbf{y} \in \mathbb{R}^{nd}. \tag{3.2.2}$$

3.2.1 Governing equations for steady-state response

We shall denote the steady-state concentration field by $c(\mathbf{x})$. The governing equations can be written as follows:

$$-\mathrm{div}[\mathbf{D}(\mathbf{x})\mathrm{grad}[c]] = f(\mathbf{x}) \qquad \text{in } \Omega, \tag{3.2.3a}$$

$$c(\mathbf{x}) = c^{\mathrm{p}}(\mathbf{x}) \qquad \text{on } \Gamma^{\mathrm{D}}, \quad \text{and} \tag{3.2.3b}$$

$$-\widehat{\mathbf{n}}(\mathbf{x}) \cdot \mathbf{D}(\mathbf{x})\mathrm{grad}[c] = q^{\mathrm{p}}(\mathbf{x}) \qquad \text{on } \Gamma^{\mathrm{N}}, \tag{3.2.3c}$$

where $f(\mathbf{x})$ is the volumetric source/sink, $c^{\mathrm{p}}(\mathbf{x})$ is the prescribed concentration, and $q^{\mathrm{p}}(\mathbf{x})$ is the prescribed flux. For uniqueness, we assume $\Gamma^{\mathrm{D}} \neq \emptyset$.

Maximum principle and the non-negative constraint

The above boundary value problem is a self-adjoint second-order elliptic partial differential equation (PDE). It is well-known that such PDEs possess an important mathematical property – the classical maximum principle [49]. The mathematical statement of the classical maximum principle can be written as follows: If $c(\mathbf{x}) \in C^2(\Omega) \cap C^0(\overline{\Omega})$, $\partial\Omega = \Gamma^{\mathrm{D}}$, and $f(\mathbf{x}) \leq 0$ in Ω then

$$\max_{\mathbf{x}\in\overline{\Omega}} c(\mathbf{x}) = \max_{\mathbf{x}\in\partial\Omega} c^{\mathrm{p}}(\mathbf{x}). \tag{3.2.4}$$

Similarly, if $f(\mathbf{x}) \geq 0$ in Ω then

$$\min_{\mathbf{x}\in\overline{\Omega}} c(\mathbf{x}) = \min_{\mathbf{x}\in\partial\Omega} c^{\mathrm{p}}(\mathbf{x}). \tag{3.2.5}$$

To make our presentation on maximum principles simple, we have assumed stronger regularity on the solution (i.e., $c(\mathbf{x}) \in C^2 \cap C^0(\overline{\Omega})$), and assumed that Dirichlet boundary conditions are prescribed on the entire boundary. However, maximum principles requiring milder regularity conditions on the solution, even for the case when Neumann boundary conditions are prescribed on the boundary, can be found in the literature (see [121, 120]).

If $f(\mathbf{x}) \geq 0$ in Ω and $c^{\mathrm{p}}(\mathbf{x}) \geq 0$ on the entire $\partial\Omega$ then the maximum principle implies that $c(\mathbf{x}) \geq 0$ in the entire domain, which is the non-negativity of the concentration field.

Single-field Galerkin weak formulation

The following function spaces will be used in the rest of this chapter:

$$\mathcal{U} := \left\{ c(\mathbf{x}) \in H^1(\Omega) \;\middle|\; c(\mathbf{x}) = c^{\mathrm{p}}(\mathbf{x}) \text{ on } \Gamma^{\mathrm{D}} \right\} \quad \text{and} \tag{3.2.6}$$

$$\mathcal{W} := \left\{ w(\mathbf{x}) \in H^1(\Omega) \;\middle|\; w(\mathbf{x}) = 0 \text{ on } \Gamma^{\mathrm{D}} \right\}, \tag{3.2.7}$$

where $H^1(\Omega)$ is a standard Sobolev space [2]. The single-field Galerkin weak formulation corresponding to equations (3.2.3a)–(3.2.3c) reads: Find $c(\mathbf{x}) \in \mathcal{U}$ such that we have

$$\mathcal{B}(w; c) = L(w) \quad \forall w(\mathbf{x}) \in \mathcal{W}, \tag{3.2.8}$$

where the bilinear form and linear functional are, respectively, defined as

$$\mathcal{B}(w; c) := \int_{\Omega} \mathrm{grad}[w(\mathbf{x})] \cdot \mathbf{D}(\mathbf{x}) \mathrm{grad}[c(\mathbf{x})] \; \mathrm{d}\Omega \quad \text{and} \tag{3.2.9a}$$

$$L(w) := \int_{\Omega} w(\mathbf{x}) f(\mathbf{x}) \; \mathrm{d}\Omega + \int_{\Gamma^{\mathrm{N}}} w(\mathbf{x}) q^{\mathrm{p}}(\mathbf{x}) \; \mathrm{d}\Gamma. \tag{3.2.9b}$$

Since $\mathbf{D}(\mathbf{x})$ is symmetric, by Vainberg's theorem [75], the single-field Galerkin weak formulation given by equation (3.2.8) is equivalent to the following variational problem:

$$\underset{c(\mathbf{x}) \in \mathcal{U}}{\text{minimize}} \quad \frac{1}{2} \mathcal{B}(c; c) - L(c). \tag{3.2.10}$$

A methodology to enforce the maximum principle for steady-state problems

Our methodology is based on the finite element method. We decompose the domain into "$Nele$" non-overlapping open element sub-domains such that

$$\overline{\Omega} = \bigcup_{e=1}^{Nele} \overline{\Omega}^e. \tag{3.2.11}$$

(Recall that a superposed bar denotes the set closure.) The boundary of Ω^e is denoted by $\partial\Omega^e := \overline{\Omega}^e - \Omega^e$. We shall define the following finite dimensional vector spaces of $\mathcal{U}$ and $\mathcal{W}$:

$$\mathcal{U}^h := \left\{ c^h(\mathbf{x}) \in \mathcal{U} \;\middle|\; c^h(\mathbf{x}) \in C^0(\overline{\Omega}), c^h(\mathbf{x})\Big|_{\Omega^e} \in \mathbb{P}^k(\Omega^e), e = 1, \cdots, Nele \right\} \quad \text{and} \tag{3.2.12a}$$

$$\mathcal{W}^h := \left\{ w^h(\mathbf{x}) \in \mathcal{W} \;\middle|\; w^h(\mathbf{x}) \in C^0(\overline{\Omega}), w^h(\mathbf{x})\Big|_{\Omega^e} \in \mathbb{P}^k(\Omega^e), e = 1, \cdots, Nele \right\}, \tag{3.2.12b}$$

where k is a non-negative integer, and $\mathbb{P}^k(\Omega^e)$ denotes the linear vector space spanned by polynomials up to k-th order defined on the sub-domain Ω^e. The finite element formulation for equation (3.2.8) can be written as: Find $c^h(\mathbf{x}) \in \mathcal{P}^h$ such that we have

$$\mathcal{B}(q^h; c^h) = L(q^h) \quad \forall q^h(\mathbf{x}) \in \mathcal{Q}^h. \tag{3.2.13}$$

It has been documented in the literature that the above finite element formulation violates the maximum principle and the non-negative constraint [105, 132, 127].

We now outline an optimization-based methodology that satisfies the maximum principle and the non-negative constraint on general computational grids. To this end, we shall use the symbols $\preceq$ and $\succeq$ to denote component-wise inequalities for vectors. That is, for any two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$

$$\boldsymbol{a} \preceq \boldsymbol{b} \quad \text{means that} \quad a_i \leq b_i \; \forall i. \tag{3.2.14}$$

The symbol $\succeq$ can be similarly defined. Let $< \cdot ; \cdot >$ denote the standard inner-product in Euclidean space. After finite element discretization, the discrete equations corresponding to equation (3.2.13) take the form

$$\boldsymbol{K}\boldsymbol{c} = \boldsymbol{f}, \tag{3.2.15}$$

where $\boldsymbol{K}$ is a symmetric positive definite matrix, $\boldsymbol{c}$ is the vector containing nodal concentrations, and $\boldsymbol{f}$ is the force vector. Equation (3.2.15) is equivalent to the following minimization problem

$$\underset{\boldsymbol{c} \in \mathbb{R}^{ndofs}}{\text{minimize}} \quad \frac{1}{2}\langle \boldsymbol{c}; \boldsymbol{K}\boldsymbol{c} \rangle - \langle \boldsymbol{c}; \boldsymbol{f} \rangle, \tag{3.2.16}$$

where "$ndofs$" denotes the number of degrees of freedom for the nodal concentrations. Equation (3.2.15) can lead to unphysical negative solutions.

Following [127, 105], a methodology corresponding to equation (3.2.16) that satisfies the non-negative constraint can be written as follows:

$$\underset{\boldsymbol{c} \in \mathbb{R}^{ndofs}}{\text{minimize}} \quad \frac{1}{2} < \boldsymbol{c}; \boldsymbol{K}\boldsymbol{c} > - < \boldsymbol{c}; \boldsymbol{f} > \quad \text{and} \tag{3.2.17a}$$

$$\text{subject to} \quad \boldsymbol{0} \preceq \boldsymbol{c}, \tag{3.2.17b}$$

where $\boldsymbol{0}$ is a vector of size $ndofs$ containing zeros. Since $\boldsymbol{K}$ is positive definite, equation (3.2.17) has a unique global minimum [18]. Several robust numerical methods can be used to solve equation (3.2.17), which include e.g., active set strategy and interior point methods [18]. In this thesis, the optimization problems are solved using the parallel optimization toolkit TAO [123], which use the active-set Newton trust region (TRON) and quasi-Newton-based bounded limited memory variable metric (BLMVM) algorithms.

3.2.2 Governing equations for transient response

We denote the time by $t \in [0, \mathcal{I}]$, where $\mathcal{I}$ denotes the length of the time interval of interest. We denote the time-dependent concentration by $c(\mathbf{x}, t)$. The initial boundary value problem can be written as follows:

$$\frac{\partial c}{\partial t} = \mathrm{div}[\mathbf{D}(\mathbf{x})\mathrm{grad}[c]] + f(\mathbf{x}, t) \quad \text{in } \Omega \times (0, \mathcal{I}), \tag{3.2.18a}$$

$$c(\mathbf{x}, t) = c^{\mathrm{p}}(\mathbf{x}, t) \quad \text{on } \Gamma^{\mathrm{D}} \times (0, \mathcal{I}), \tag{3.2.18b}$$

$$-\widehat{\mathbf{n}}(\mathbf{x}) \cdot \mathbf{D}(\mathbf{x})\mathrm{grad}[c] = q^{\mathrm{p}}(\mathbf{x}, t) \quad \text{on } \Gamma^{\mathrm{N}} \times (0, \mathcal{I}), \quad \text{and} \tag{3.2.18c}$$

$$c(\mathbf{x}, 0) = c_0(\mathbf{x}) \quad \text{in } \Omega, \tag{3.2.18d}$$

where $c_0(\mathbf{x})$ is the prescribed initial concentration, $f(\mathbf{x}, t)$ is the time-dependent volumetric source/sink, $c^{\mathrm{p}}(\mathbf{x}, t)$ is the time-dependent prescribed concentration on the boundary, and $q^{\mathrm{p}}(\mathbf{x}, t)$ is the prescribed time-dependent flux on the boundary.

The maximum principle and the non-negative constraint

The maximum principle of a transient diffusion equation asserts that the maximum can occur only on the boundary of the domain or in the initial condition if $f(\mathbf{x}, t) \leq 0$ and $\Gamma^{\mathrm{D}} = \partial\Omega$. Mathematically, a solution to equations (3.2.18a)–(3.2.18d) will satisfy:

$$c(\mathbf{x}, t) \leq \max\left[\max_{\mathbf{x}\in\Omega} c_0(\mathbf{x}), \max_{\mathbf{x}\in\partial\Omega} c^{\mathrm{p}}(\mathbf{x}, t)\right] \quad \forall t, \tag{3.2.19}$$

provided $f(\mathbf{x},t) \leq 0$. Similarly, the minimum will occur either on the boundary or in the initial condition if $f(\mathbf{x},t) \geq 0$. That is, if $f(\mathbf{x},t) \geq 0$ then a solution to equations (3.2.18a)–(3.2.18a) satisfies

$$c(\mathbf{x},t) \geq \min \left[\min_{\mathbf{x}\in\Omega} c_0(\mathbf{x}), \min_{\mathbf{x}\in\partial\Omega} c^{\mathrm{p}}(\mathbf{x},t) \right] \quad \forall t. \tag{3.2.20}$$

If $f(\mathbf{x},t) \geq 0$ in Ω, $c^{\mathrm{p}}(\mathbf{x},t) \geq 0$ on the entire $\partial\Omega$, and $c_0(\mathbf{x}) \geq 0$ in Ω then the maximum principle implies that $c(\mathbf{x},t) \geq 0$ in the entire domain at all times, which is the non-negative constraint for the concentration field for transient problems.

A methodology to enforce the maximum principle for transient problems

We divide the time interval of interest into $\mathcal{N}$ sub-intervals. That is,

$$[0,\mathcal{I}] := \bigcup_{n=0}^{\mathcal{N}} [t_n, t_{n+1}], \tag{3.2.21}$$

where t_n denotes the n-th time-step. We assume that the time-step is uniform, which can be written as

$$\Delta t = t_{n+1} - t_n. \tag{3.2.22}$$

Following the recommendation provided in [131] to meet maximum principles, we employ the backward Euler method for temporal discretization. We shall denote the nodal concentrations at the n-th time-step by $\boldsymbol{c}^{(n)}$. We shall denote the minimum and maximum values for the concentration by $c_{\min}$ and $c_{\max}$, which will be provided by the maximum principle and the non-negative constraint. At each time-step, one has to solve the following convex quadratic program:

$$\underset{\boldsymbol{c}^{(n+1)}}{\text{minimize}} \quad \frac{1}{2}\langle \boldsymbol{c}^{(n+1)}; \widetilde{\boldsymbol{K}}\boldsymbol{c}^{(n+1)}\rangle - \langle \boldsymbol{c}^{(n+1)}; \widetilde{\boldsymbol{f}}^{(n+1)}\rangle \quad \text{and} \tag{3.2.23a}$$

$$\text{subject to} \quad c_{\min}\mathbf{1} \preceq \boldsymbol{c}^{(n+1)} \preceq c_{\max}\mathbf{1}, \tag{3.2.23b}$$

where

$$\widetilde{\boldsymbol{K}} := \frac{1}{\Delta t}\boldsymbol{M} + \boldsymbol{K} \qquad \text{and} \tag{3.2.24}$$

$$\widetilde{\boldsymbol{f}}^{(n+1)} := \boldsymbol{f}^{(n+1)} + \frac{1}{\Delta t}\boldsymbol{M}\boldsymbol{c}^{(n+1)}. \tag{3.2.25}$$

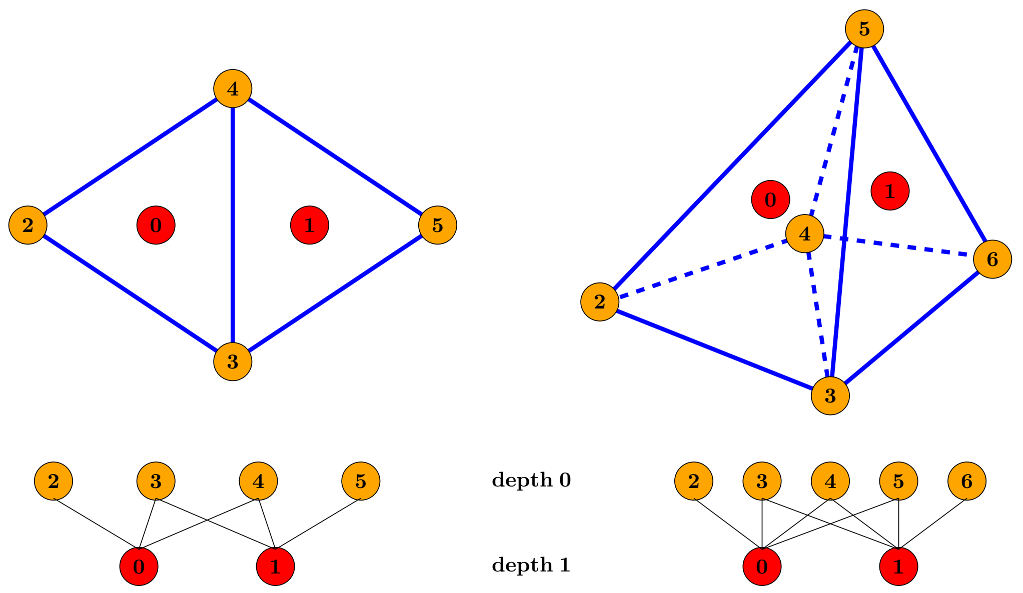
In the above equation, $\boldsymbol{M}$ is the capacity matrix [131].

3.3 PARALLEL IMPLEMENTATION

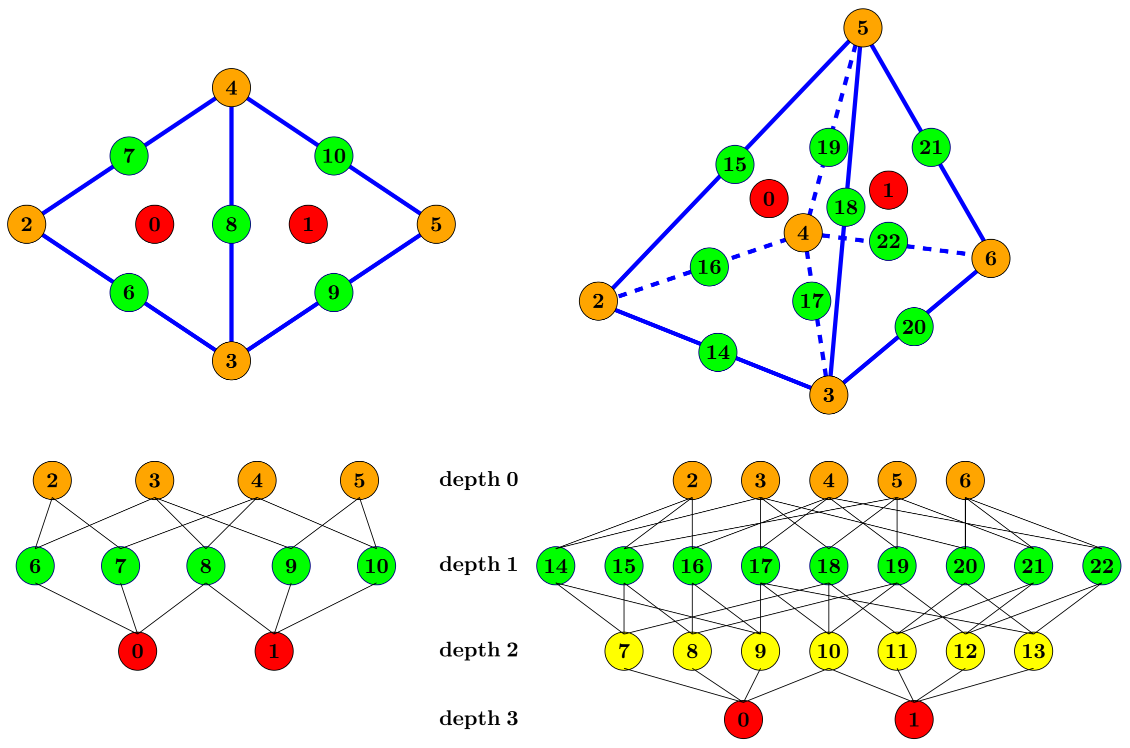
3.3.1 PETSc and TAO

We leverage existing scientific libraries such as PETSc and TAO to formulate our computational framework for large-scale problems. PETSc is a suite of data structures and routines for the parallel solution of scientific applications. It also provides interfaces to several other libraries such as Metis/ParMETIS[80] and HDF5 [158] for mesh partitioning and binary data format handling respectively. The Data Management (DM) data structure is used to manage all information including vectors and sparse matrices and compatible with binary data formats. To handle unstructured grids in parallel, a subset of the DM structure called DMPlex (see [92, 89, 12]), as shown in Figure 3.1, uses direct acyclic graphs to organize all mesh information. This topology enables the freedom to mix and match various non vertex-based discretization methods, such as the two-point flux finite volume method, and the classical mixed formulations based on the lowest-order Raviart Thomas finite element space.

Another important feature within PETSc is TAO. The TAO library has a suite



(a) Optimal 2D and 3D elements



(b) Interpolated 2D and 3D elements

Figure 3.1: Representation of mesh points within the DMPlex data structure and their associated directed acyclic graphs.

of data structures and routines that enable the solution of large-scale optimization problems. It can support any data structure or solver within PETSc. Our DMP methodology will use both the TRON and BLMVM solvers available within TAO. BLMVM is a quasi-Newton method that uses projected gradients to approximate the Hessian, which is useful for problems where the Hessian is too complicated or expensive to compute. Other optimization algorithms such as TRON and the Gradient Projected Conjugate Gradient (GPCG) typically require Hessian information and more memory, but they are expected to converge more rapidly than BLMVM. Further details regarding the implementation of these various methods may be found in [123] and the references therein.

3.3.2 Finite element implementation

PETSc abstractions for finite elements, quadrature rules, and function spaces have also been recently introduced and are suitable for the mesh topology within DMPlex. They are built upon the same framework as the FInite element Automatic Tabulator (FIAT) found within the FEniCS Project [85, 107, 108]. The finite element discretizations simply need the equations, auxiliary coefficients (e.g., permeability, diffusivity, etc.), and boundary conditions specified as point-wise functions. We express all discretizations in nonlinear form so let $\boldsymbol{r}$ and $\boldsymbol{J}$ denote the residual and Jacobian respectively.

Following the FEM model outlined in [88], we consider the weak form that depends on fields and gradients. The residual evaluation can be expressed as

$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{r}(\boldsymbol{c}) \sim \int_{\Omega^e} \left[w \cdot \mathcal{F}_0\left(c, \nabla c\right) + \nabla w \cdot \boldsymbol{\mathcal{F}}_1(c, \nabla c)\right] \mathrm{d}\Omega = 0, \tag{3.3.1}$$

where $\mathcal{F}_0(c, \nabla c)$ and $\boldsymbol{\mathcal{F}}_1(c, \nabla c)$ are user-defined point-wise functions that capture the problem physics. This framework decouples the problem specification from the mesh and degree of freedom traversal. That is, the scientist need only focus on providing point function evaluations while letting the finite element library take care of meshing, quadrature points, basis function evaluation, and mixed forms if any. The discretization of the residual is written as

$$\boldsymbol{r}(\boldsymbol{c}) = \mathop{\mathbf{\mathsf{A}}}_{e=1}^{Nele} \begin{bmatrix} \boldsymbol{N}^{\mathrm{T}} & \boldsymbol{B}^{\mathrm{T}} \end{bmatrix} \boldsymbol{W} \begin{bmatrix} \mathcal{F}_0(c_q, \nabla c_q) \\ \boldsymbol{\mathcal{F}}_1(c_q, \nabla c_q) \end{bmatrix}, \tag{3.3.2}$$

where $\mathbf{\mathsf{A}}$ represents the standard assembly operator, $\boldsymbol{N}$ and $\boldsymbol{B}$ are matrix forms of basis functions that reduce over quadrature points, $\boldsymbol{W}$ is a diagonal matrix of quadrature weights (including the geometric Jacobian determinant of the element), and c_q is the field value at quadrature point q. The Jacobian of (3.3.2) needs only the derivatives of the point-wise functions

$$\boldsymbol{J}(\boldsymbol{c}) = \mathop{\mathbf{\mathsf{A}}}_{e=1}^{Nele} \begin{bmatrix} \boldsymbol{N}^{\mathrm{T}} & \boldsymbol{B}^{\mathrm{T}} \end{bmatrix} \boldsymbol{W} \begin{bmatrix} \mathcal{F}_{0,0} & \mathcal{F}_{0,1} \\ \boldsymbol{\mathcal{F}}_{1,0} & \boldsymbol{\mathcal{F}}_{1,1} \end{bmatrix} \begin{bmatrix} \boldsymbol{N} \\ \boldsymbol{B} \end{bmatrix} \quad \text{and} \tag{3.3.3a}$$

$$[\mathcal{F}_{i,j}] = \begin{bmatrix} \frac{\partial \mathcal{F}_0}{\partial c} & \frac{\partial \mathcal{F}_0}{\partial \nabla c} \\ \frac{\partial \boldsymbol{\mathcal{F}}_1}{\partial c} & \frac{\partial \boldsymbol{\mathcal{F}}_1}{\partial \nabla c} \end{bmatrix} (c_q, \nabla c_q), \tag{3.3.3b}$$

The point-wise functions corresponding to the weak form in (3.2.9) would be

$$\mathcal{F}_0 = -f(\mathbf{x}), \quad \boldsymbol{\mathcal{F}}_1 = \mathbf{D}(\mathbf{x}) \nabla c_q \quad \text{and} \tag{3.3.4a}$$

$$\mathcal{F}_{0,0} = 0, \quad \mathcal{F}_{0,1} = 0, \quad \boldsymbol{\mathcal{F}}_{1,0} = \mathbf{0}, \quad \boldsymbol{\mathcal{F}}_{1,1} = \mathbf{D}(\mathbf{x}). \tag{3.3.4b}$$

Similarly, the point-wise functions for the transient response are

$$\mathcal{F}_0 = \dot{c}_q - f(\mathbf{x}, t), \quad \boldsymbol{\mathcal{F}}_1 = \mathbf{D}(\mathbf{x}) \nabla c_q \quad \text{and} \tag{3.3.5a}$$

$$\mathcal{F}_{0,0} = \frac{1}{\Delta t}, \quad \mathcal{F}_{0,1} = 0, \quad \boldsymbol{\mathcal{F}}_{1,0} = \mathbf{0}, \quad \boldsymbol{\mathcal{F}}_{1,1} = \mathbf{D}(\mathbf{x}), \tag{3.3.5b}$$

Algorithm 1 Pseudocode for the large-scale transport solver

```
Create/input DAG on rank 0.
Create/input cell-wise velocity on rank 0.
if size > 1 then
    Partition mesh among all processors.
end if
Refine distributed mesh if necessary.
Create PetscSection and FE discretization.
Set $n = 0$ and $\boldsymbol{c}^{(0)} = 10^{-8}$.
Insert Dirichlet BC constraints into $\boldsymbol{c}^{(0)}$.
Compute Jacobian $\boldsymbol{J}$.
while true do                                  ▷ Begin time-stepping scheme.
    Compute Residual $\boldsymbol{r}^{(n)}$ .
    if Classical Galerkin then                 ▷ Solve without non-negative methodology.
        $\boldsymbol{c}^{(n+1)} = \boldsymbol{c}^{(n)} - \boldsymbol{J} \backslash \boldsymbol{r}^{(n)}$.
    else                                       ▷ Solve with non-negative methodology.
        TaoSolve() for $\boldsymbol{c}^{(n+1)}$ based on equations (3.3.8) and (3.3.9).
    end if
    if steady-state or $(n)$ == total number of time steps then
        break.
    else
        $n+ = 1$.
    end if
end while
```

where $\dot{c}_q$ denotes the time derivative. A similar discretization is used to project the Neumann boundary conditions into the residual vector. Assuming a fixed time-step, $[\mathcal{F}_{i,j}]$ and the Jacobian in equation (3.3.3b) do not change with time and have to be computed only once. If n denotes the time step ($n = 0$ denotes the initial condition) then the residual and Jacobian can be defined as

$$\boldsymbol{r}^{(n)} \equiv \boldsymbol{r}(\boldsymbol{c}^{(n)}) \quad \text{and} \tag{3.3.6}$$

$$\boldsymbol{J} \equiv \boldsymbol{J}(\boldsymbol{c}^{(0)}). \tag{3.3.7}$$

To enforce the non-negative methodology, the following objective function b and gradient function $\boldsymbol{g}$ is provided

$$b = \frac{1}{2}\boldsymbol{c}^{(n+1)} \cdot \boldsymbol{J}\boldsymbol{c}^{(n+1)} + \boldsymbol{c}^{(n+1)} \cdot \left[\boldsymbol{r}^{(n)} - \boldsymbol{J}\boldsymbol{c}^{(n)}\right] \quad \text{and} \tag{3.3.8}$$

$$\boldsymbol{g} = \boldsymbol{J}\left[\boldsymbol{c}^{(n+1)} - \boldsymbol{c}^{(n)}\right] + \boldsymbol{r}^{(n)}. \tag{3.3.9}$$

BLMVM relies only on the above two equations, whereas TRON needs the Hessian which is equivalent to $\boldsymbol{J}$. Algorithm 1 outlines the steps taken in our computational framework.

3.4 PERFORMANCE MODELING

PETSc is a constantly evolving open-source library that brings out new features and algorithms almost every day. It has capabilities to interface with a large number of other open-source application, software, and linear algebra packages. However, it is not always known which algorithms or software packages will have the best performance on an HPC system, especially if these packages have little documentation and are used as black-box solvers. Computational scientists would like to know which solvers or algorithms to use for their specific need before using an HPC system. The simple metric in answering this question is the time-to-solution. However, in order to gain insight into whether an observed time-to-solution is good, information about hardware utilization and algorithmic efficiency is needed. Further potential scalability issues as a function of problem size and computational resources are often needed.

Hardware specifications of HPC systems and software implementations significantly impact the performance of numerical algorithms. Ideally we want simulations to consume as little wall-clock time as possible for the resources used. Several factors such as compiler optimizations, cache policies, locality of reference, and code implementation may drastically affect the performance. Table 3.1 lists the hardware

Table 3.1: List of HPC systems used in this study

	Mustang (MU)	Wolf (WF)
Processor	AMD Opteron 6176	Intel Xeon E5-2670
Clock rate	2.3 GHz	2.6 GHz
FLOPs/cycle/core	4	8
Sockets per compute node	2	2
NUMA nodes per socket	2	1
Cores per socket	12	8
Total cores (compute nodes)	38400 (1600)	9856 (616)
Memory per compute node	64 GB	64 GB
L1 cache per core	64 KB	32 KB
L2 cache per core	512 KB	256 KB
L3 cache per socket	12 MB	20 MB
Interconnect	40 Gb/s	40 Gb/s

specifications of the two HPC systems (Mustang and Wolf) that are used in our numerical experiments in this chapter. The Mustang HPC system consists of relatively old processors. One could simply measure wall-clock time across multiple compute nodes on the respective HPC systems and determine the parallel efficiency of a certain algorithm, but we are interested in quantifying how different algorithms behave sequentially and what kind of parallel performance to expect before running numerical simulations. The wall-clock time of any simulation can generally be summed up as a function of three things: the workload measured as arithmetic operations, transfer of data between the memory and CPU registers, and internode communication. Hardware efficiency is traditionally measured as the ratio of time taken if the arithmetic capability is used to 100% and actual time taken.

For many applications, the memory bandwidth is the performance limiting factor on common computing platforms. That is, the floating point performance (FLOPS/s) of several application codes will never reach the theoretical peak performance (TPP) determined based on the aggregate peak core rate assuming 100% utilization. This limitation is particularly important for iterative solvers and optimization methods that rely on numerous sparse matrix-vector (SpMV) multiplications (see [113]

Table 3.2: Commonly used PETSc operations and their respective TBT. Here we note X, Y, Z as vectors with $i = 1, \cdots, N$ entries, a is a scalar value, and nz denotes the total number of non-zeros, with all variables being in 8 byte precision.

PETSc function	Operation	Total Bytes Transferred
VecNorm()	$a = \sqrt{\sum_i^N X(i)^2}$	$8(N+1)$
VecDot()	$a = \sum_i^N X(i) * Y(i)$	$8(2N+1)$
VecCopy()	$Y \leftarrow X$	$8(2N)$
VecSet()	$Y(i) = a$	$8(2N)$
VecScale()	$Y = a * Y$	$8(2N)$
VecAXPY()	$Y = a * X + Y$	$8(3N)$
VecAYPX()	$Y = X + a * Y$	$8(3N)$
VecPointwiseMult()	$Z(i) = X(i) * Y(i)$	$8(3N)$
MatMult()	SpMV	$4(N + nz) + 8(2N + nz)$

and the references within). The frequent use of SpMV allows for little cache reuse and will result in a large number of cache misses that are expensive in regards to performance. Such behavior is important to understand when determining how efficient a scientific code is. Performance models such as the Roofline Model [164, 106] include impact of memory transfers into the performance model to better quantify performance limitations due to hardware properties. Performance models can help application developers identify bottlenecks and indicate which areas of the code can be further optimized. In the next section, we will demonstrate that such models can also be used to predict the parallel efficiency of various optimization solvers on two different LANL HPC systems. The key parameter for these models is the Arithmetic Intensity (AI) which we recall is defined as total FLOP over Total Bytes Transferred (TBT). The AI in combination with the memory bandwidth and the TPP creates a "roofline" for the estimation of ideal peak performance. In Chapter 2, the TBT was based on the total number of cache misses, but in this chapter, we will use a different cache model to define the TBT.

To this end, we propose a roofline-like performance model where the TBT assumes a "perfect cache" – each byte of the data needs to be fetched from DRAM

(Dynamic Random Access Memory) only once, and every cache line is freely allocated. Table 3.2 lists the key PETSc functions used for the solvers and their respective estimates of TBT based on the perfect cache assumption. The formula for SpMV follows the procedure outlined in [65]. We assume that the TBT formula for operations also involving a sparse matrix and vector, like the incomplete lower-upper (ILU) factorization, to be the same as MatMult(). Estimating the TBT for other important operations like the sparse matrix-matrix and triple matrix products (which are important for multi-grid methods) is an area of future work. In short, our AI formulation relies on the following four key assumptions:

(i) All floating-point operations (add, multiply, etc.) are treated equally and equate to one FLOP count.

(ii) There are no cache conflict misses, and cache lines are fully utilized. That is, each matrix and vector element is loaded into cache only once.

(iii) Processor never waits on a memory reference. That is, every load and store operation is satisfied in a single cycle.

(iv) Compilers are capable of storing scalar multipliers in the register for streaming computations.

Therefore, the efficiency based on this new roofline-like performance model is written as

$$\text{Roof-line Efficiency (\%)} = \frac{\text{Measured FLOPS/s}}{\min\begin{cases}\text{TPP} \\ \text{AI} \times \text{“perfect cache” bandwidth}\end{cases}} \times 100, \quad (3.4.1)$$

For FLOP/s we use the number reported by the PETSc program. The denominator is the ideal performance upper-bounded based either on the TPP or the product of the

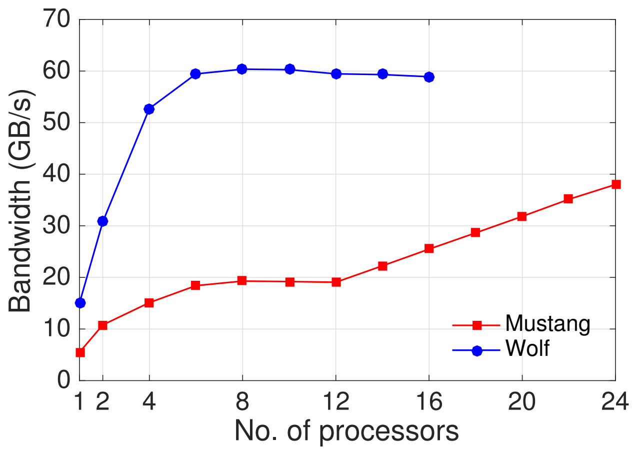


Figure 3.2: Measured memory bandwidth of a single Mustang and Wolf compute node based on the STREAMS Triad Benchmark.

AI, based on perfect cache assumption, and bandwidth measured by the STREAM Triad benchmark [114]. Figure 3.2 denotes the estimated memory bandwidth as a function of number of MPI processes on a single Mustang and Wolf node with MPI processes allocated in a round-robin fashion between CPUs in a node. It is interesting to note that although the Wolf node has a higher STREAM bandwidth, there is no performance gain past eight cores. This means that using a Wolf compute node for memory-bandwidth bound applications, no performance gain can be expected beyond eight cores, whereas one would still see some performance gains when using all 24 cores on a Mustang node.

The performance model based on equation (3.4.1) is a serial model. For the Mustang and Wolf systems, it can be seen from Figure 3.2 that the single core measured bandwidth is 5.65 GB/s and 15.5 GB/s respectively. It should be noted that this performance model does account for cache effects. That is, it does not quantify the useful bandwidth sustained for some particular level of cache. The true hardware and algorithmic efficiency is not reflected by this model, so our aim is to show relative performance between select PETSc and TAO solvers. Comparing the AI based on the measured FLOPS/s and STREAM bandwidths will give us a better understanding of how high-performing the PETSc and TAO solvers are for select problems.

3.5 REPRESENTATIVE NUMERICAL RESULTS

In this section, we compare the performance of our non-negative methodology using the TAO solver to that of the Galerkin formulation using the Krylov Subspace (KSP) solver. We examine the performance using two problems:

(i) a unit cube with a hole under steady-state, and

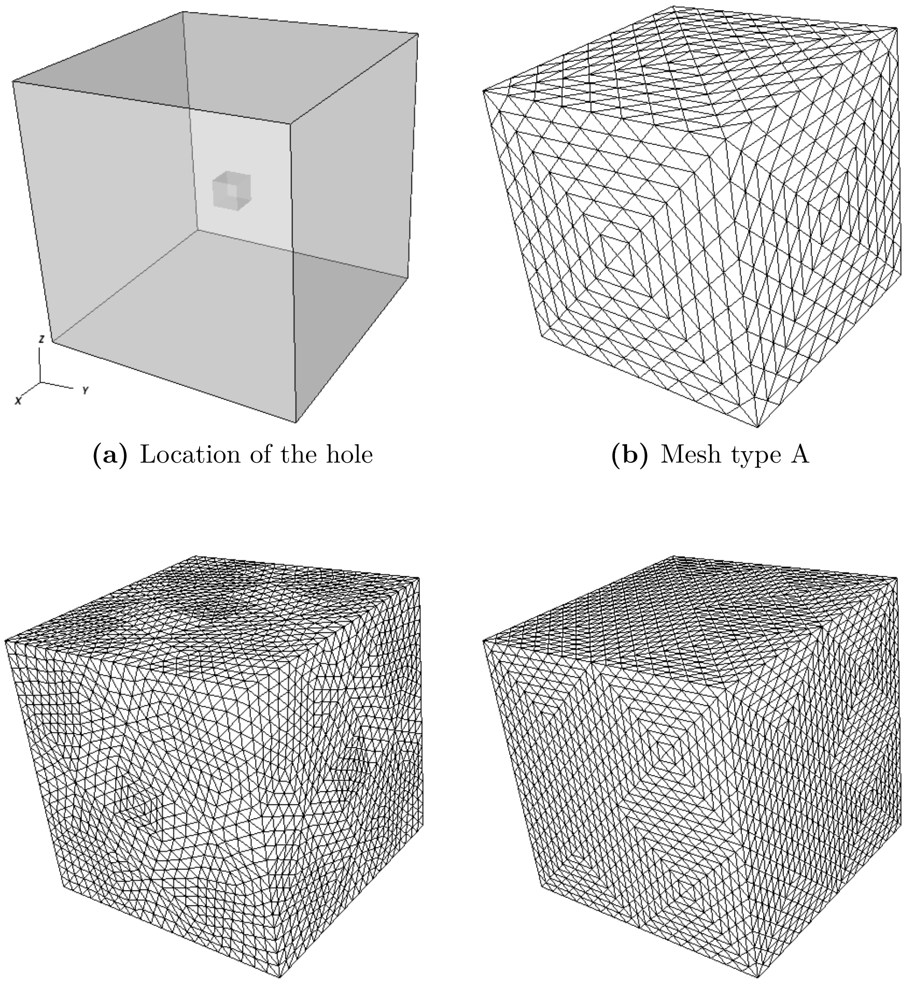
(ii) a transient Chromium transport problem.

The diffusivity tensor is assumed to depend on the flow velocity through

$$\mathbf{D}(\mathbf{x}) = (\alpha_T \|\mathbf{v}\| + D_M)\,\mathbf{I} + (\alpha_L - \alpha_T)\frac{\mathbf{v} \otimes \mathbf{v}}{\|\mathbf{v}\|}, \tag{3.5.1}$$

where α_L, α_T, and D_M denote the longitudinal dispersivity, transverse dispersivity and molecular diffusivity, respectively. We employ the conjugate gradient method and the block Jacobi/ILU(0) preconditioner for solving the linear system from the Galerkin formulation and employ TAO's TRON and BLMVM methods for the non-negative methodology. The relative convergence tolerances for both KSP and TAO solvers are set to 10^{-6}, and Δt for the transient response in the Chromium problem is initially set to 0.2 days. For strong-scaling studies shown here, we used OpenMPI v1.6.5 for message passing and bind processes to cores alternating between sockets. ParaView[10] and VisIt[35] were used to generate all contour and mesh plots.

Remark 3.5.1. *Throughout this chapter, the non-negative methodology that we refer to, is in fact a discrete maximum principle preserving methodology. In that, along with the non-negative constraint we also enforce that the concentrations are less than or equal to 1.*



(a) Location of the hole **(b)** Mesh type A

(c) Mesh type B **(d)** Mesh type C

Figure 3.3: Cube with a hole: pictorial description and the associated unstructured grids.

3.5.1 Anisotropic diffusion in a unit cube with a cubic hole

Let the computational domain be a unit cube with a cubic hole of size $[4/9, 5/9] \times [4/9, 5/9] \times [4/9, 5/9]$. The concentration on the outer boundary is taken to be zero and the concentration on the interior boundary is taken to be unity. The volumetric source is taken as zero (i.e., $f(\mathbf{x}) = 0$). The velocity vector field for this problem is chosen to be

$$\mathbf{v}(\mathbf{x}) = \mathbf{e}_x + \mathbf{e}_y + \mathbf{e}_z. \tag{3.5.2}$$

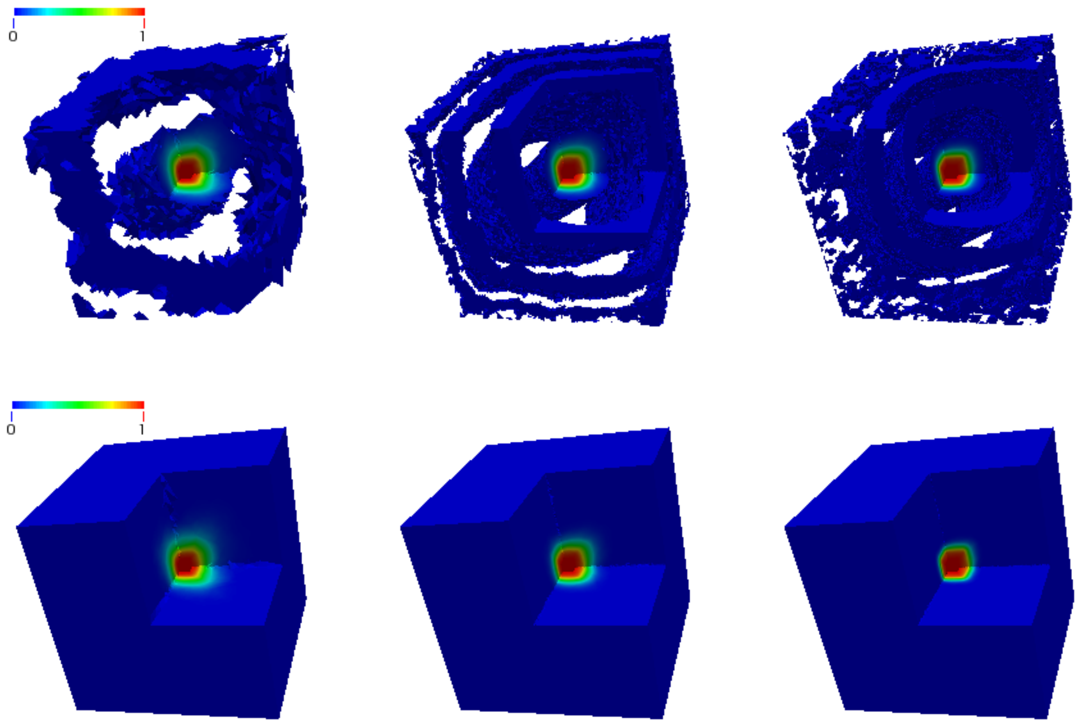


Figure 3.4: Cube with a hole: numerical solution for cases A1 (left), B2 (middle), and C3 (right) using the Galerkin formulation (top row) and non-negative methodology (bottom row).

The diffusion parameters are set as: $\alpha_L = 1$, $\alpha_T = 0.001$, and $D_M = 0$. The pictorial description of the computational domain and the three mesh types composed of 4-node tetrahedrons are shown in Figure 3.3. We consider three unstructured mesh types with three levels of element-wise mesh refinement, giving us nine total case studies of increasing problem size as shown in Table 3.3. Five different solver simulations were used for this study:

- Galerkin with CG/block Jacobi.
- TRON1: with KSP tolerance of 10^{-1}.
- TRON2: with KSP tolerance of 10^{-2}.
- TRON3: with KSP tolerance of 10^{-3}.

Table 3.3: Cube with a hole: list of various mesh type and refinement level combinations used.

Case	Mesh type	Refinement level	Tetrahedrons	Vertices
A1	A	1	199,296	36,378
B1	B	1	409,848	75,427
C1	C	1	793,824	140,190
A2	A	2	1,594,368	278,194
B2	B	2	3,278,784	574,524
C2	C	2	6,350,592	1,089,562
A3	A	3	12,754,994	2,175,330
B3	B	3	26,230,272	4,483,126
C3	C	3	50,804,736	9,172,044

Table 3.4: Cube with a hole: minimum and maximum concentrations for each case.

Case	Min. concentration	Max. concentration	% nodes violated
A1	-0.0224825	1.00000	9,518/36,378 $\rightarrow$ 26.2%
B1	-0.0139559	1.00000	32,247/43,180 $\rightarrow$ 42.8%
C1	-0.0125979	1.00000	57,272/140,190 $\rightarrow$ 40.9%
A2	-0.0311518	1.00103	82,983/278,194 $\rightarrow$ 29.2%
B2	-0.0143857	1.00000	255,640/574,524 $\rightarrow$ 44.9%
C2	-0.0119539	1.00972	453,766/1,089,562 $\rightarrow$ 41.6%
A3	-0.0258559	1.00646	643,083/2,175,330 $\rightarrow$ 29.6%
B3	-0.0115908	1.00192	2,073,934/4,483126 $\rightarrow$ 46.3%
C3	-0.0096186	1.00545	4,932,551/9,172,044 $\rightarrow$ 53.8%

- BLMVM.

The TRON solvers also use the CG and block Jacobi preconditioner but with different KSP tolerances. Numerical results for both the Galerkin formulation and the DMP methodologies for some of the mesh cases are shown in Figure 3.4. The top row of figures arise from the Galerkin formulation where the white regions denote negative concentrations, and the bottom row arise from either TRON or BLMVM. Details concerning the violation of the DMP for each case study can be found in Table 3.4. Concentrations both negative and greater than one arise for all case studies. Moreover, simply refining the mesh does not resolve these issues; in fact, refinement worsens the violation. These numerical results indicate that our computational framework can successfully enforce the DMP for diffusion problems with highly anisotropic

Table 3.5: Cube with a hole: wall-clock times (seconds) on Mustang for each solver.

Case	Galerkin	TRON1	TRON2	TRON3	BLMVM
A1	0.337	0.933	0.981	1.14	2.62
B1	0.790	1.72	2.06	2.71	5.04
C1	2.24	4.34	5.80	7.74	13.5
A2	7.21	15.2	21.7	32.5	72.0
B2	15.4	30.0	43.7	57.5	109
C2	40.4	67.8	113	118	286
A3	121	225	414	599	1167
B3	315	498	1061	1344	2524
C3	997	1539	2490	4365	9679

Table 3.6: Cube with a hole: wall-clock times (seconds) on Wolf for each solver.

Case	Galerkin	TRON1	TRON2	TRON3	BLMVM
A1	0.126	0.388	0.396	0.449	1.01
B1	0.314	0.720	0.853	1.07	2.03
C1	0.888	1.91	2.47	3.31	5.71
A2	2.58	6.34	8.74	12.8	26.2
B2	5.90	12.9	17.8	22.8	46
C2	16.2	30.1	47.3	48.9	133
A3	48.0	98.4	129	247	609
B3	107	171	342	435	1060
C3	281	467	870	1245	3131

diffusivity.

Performance modeling

We first consider the wall-clock time spent in the solvers on a single core. Tables 3.5 and 3.6 depict the solver time for each mesh, and we first note that the Mustang system requires significantly more wall-clock time to obtain a solution than Wolf; based on the TPP for the cores (9.2 GFLOPS/s for Mustang and 20.8 for Wolf), Wolf could be expected to be about 2.25 times faster, but based on STREAM, Wolf could expect to be close to three times faster if the code is memory-bandwidth limited. The wall-clock time in Tables 3.5 and 3.6 are not linearly proportional to problem size for either the Galerkin or optimization solvers. In regards to the solvers, BLMVM can require as much as ten times the amount of wall-clock time as the Galerkin method.

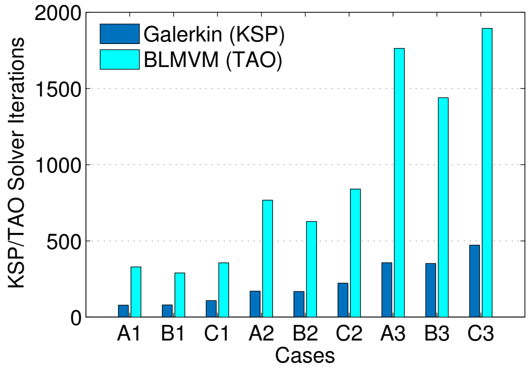


Figure 3.5: Cube with a hole: solver iterations needed for Galerkin and BLMVM.

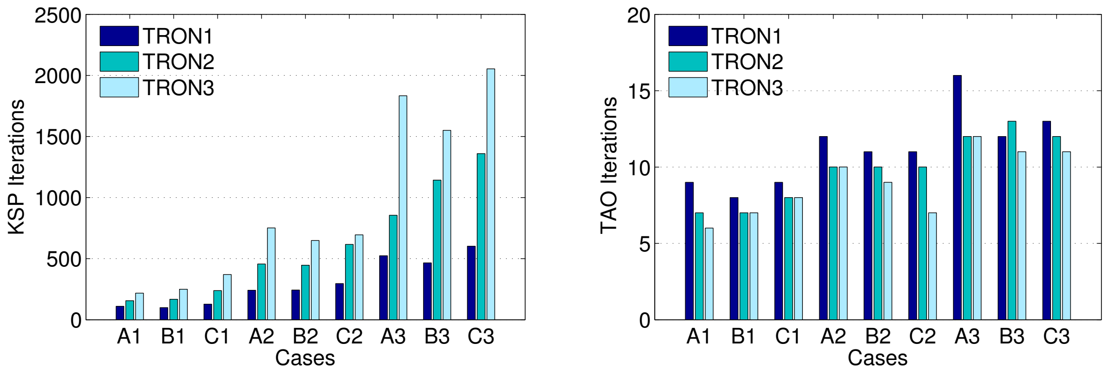
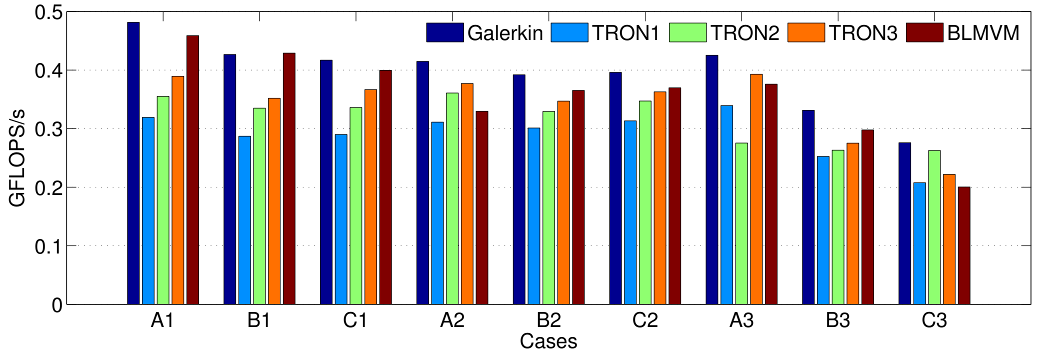
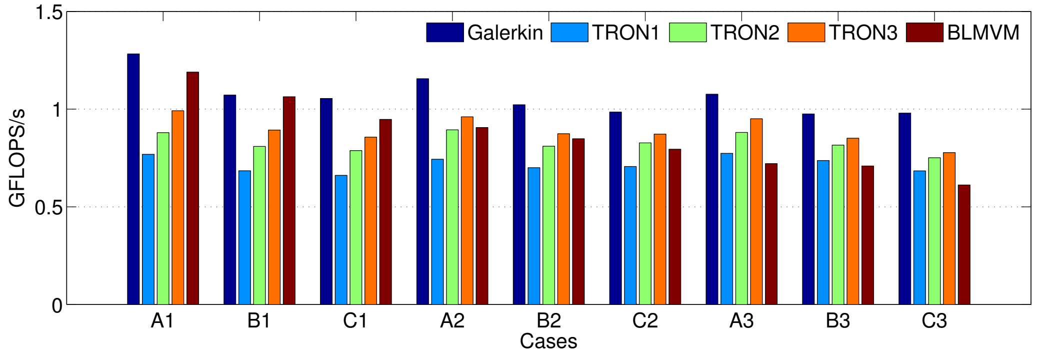


Figure 3.6: Cube with a hole: KSP (left) and TAO (right) solver iterations needed for TRON.

TRON on the other hand, does not consume nearly as much time but tightening the KSP tolerances will gradually increase the amount of time. We are interested in determining why these optimization solvers consume more wall-clock time, whether it be mostly due to additional workload associated with optimization-based techniques or less efficient use of hardware resources due to the presence of relatively more complicated data structures compared to the standard solvers used for the Galerkin formulation. The first step is noting the total KSP and TAO iterations needed and how they vary with respect to problem size. Figure 3.5 depicts the KSP and TAO iterations for the Galerkin and BLMVM methods respectively. It is well-known that block Jacobi (also known as ILU(0)) requires more iterates as the size of the problem increases. In other words, scaling up the size of the problem will potentially increase



(a) Mustang



(b) Wolf

Figure 3.7: Cube with a hole: measured floating-point rate (FLOPS/s) on a single core.

the time-to-solution needed for convergence, and we see that the BLMVM algorithm exhibits a similar rate of increase in solver iterates with respect to size. For the TRON solvers, we document both the KSP and TAO iterates as shown in Figure 3.6. We see that tightening the KSP tolerance increases the number of KSP iterates but reduces the number of TAO iterations needed. This behavior indicates that the more accurate the computed gradient projection is, the fewer optimization loops the framework needs to perform.

We also examine the measured floating-point rate provided by the PETSc performance logs, as shown in Figure 3.7, of all five solvers across their respective machines, and the floating point performance decreases as the problem size grows. The maximum reported floating point rates on the Mustang and Wolf systems are 0.47 and 1.25 GFLOPS/s, respectively. When these numbers are compared the systems' respective

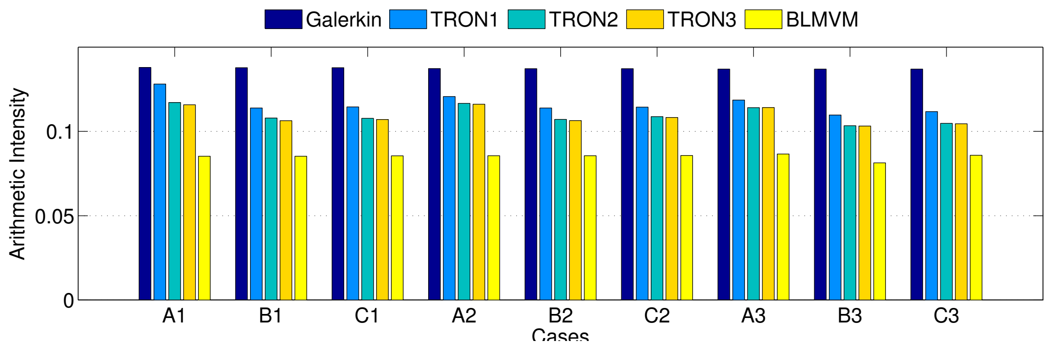
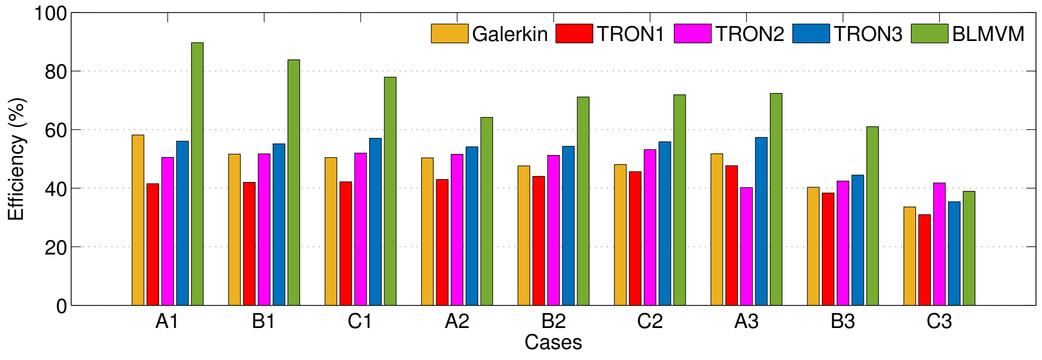
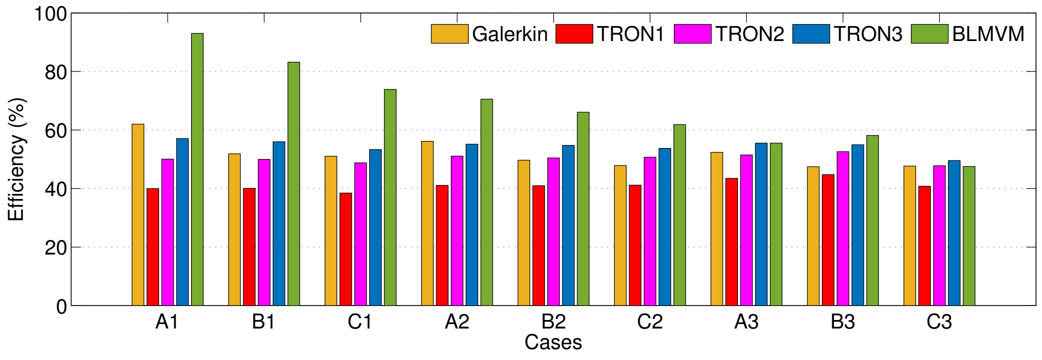


Figure 3.8: Cube with a hole: arithmetic intensity (Total FLOP over TBT) for all solvers and all cases on a single processor.

single core TPP (9.2 GFLOPS/s for Mustang and 20.8 GFLOPS/s for Wolf), the hardware efficiencies are shown to be no greater than 5.1% and 6.0% for Mustang and Wolf, respectively, and it is difficult to draw any conclusions with regard to the computational performance other than that the floating-point efficiencies are about the same for the two different architectures. The calculated AI, based on the "perfect cache" bandwidth as described in Section 3.4, is shown in Figure 3.8. It is interesting to note that the AI remains largely invariant with problem size unlike the wall-clock time, solver iterations, and floating point rates. According to the perfect cache model, the Galerkin formulation's AI is greater than any of the non-negative methodologies. Using these metrics in equation (3.4.1) as well as the STREAM Triad bandwidth of one core as shown in Figure 3.2, the estimated roofline-based efficiencies are shown in Figure 3.9. Although the raw floating-point rate of BLMVM is lower than the Galerkin method, the roofline model suggests that BLMVM is actually more efficient in a hardware sense. The TRON methods have much lower floating-point rates, but these metrics can be improved or "gamed" by tightening the KSP tolerances. This behavior leads us to believe that there is some latency associated with setting up the data structures needed to compute gradient descent projections.



(a) Mustang



(b) Wolf

Figure 3.9: Cube with a hole: roofline efficiency from equation (3.4.1) based on the measured bandwidth from STREAM.

Strong-scaling

The metric of interest is the strong-scaling potential. We conduct strong-scaling studies to measure the speedup of all nine case studies of up to 64 cores. 4 Mustang nodes with 16 of 24 cores per node and 8 Wolf nodes with 8 of 16 cores per node are allocated for this study. Figure 3.10 depicts the speedup on the Mustang system, and Figure 3.11 depicts the speedup on the Wolf system. The results indicate that Wolf exhibits better strong-scaling. For all problems and machines, the TRON simulations are slightly less efficient in the parallel sense but can be improved by tightening the KSP tolerances. Interestingly, the BLMVM algorithm not only has the best roofline efficiency but also the best parallel speedup. The super linear speedup observed for BLMVM on Wolf is likely due to diminishing cache misses when the workload per

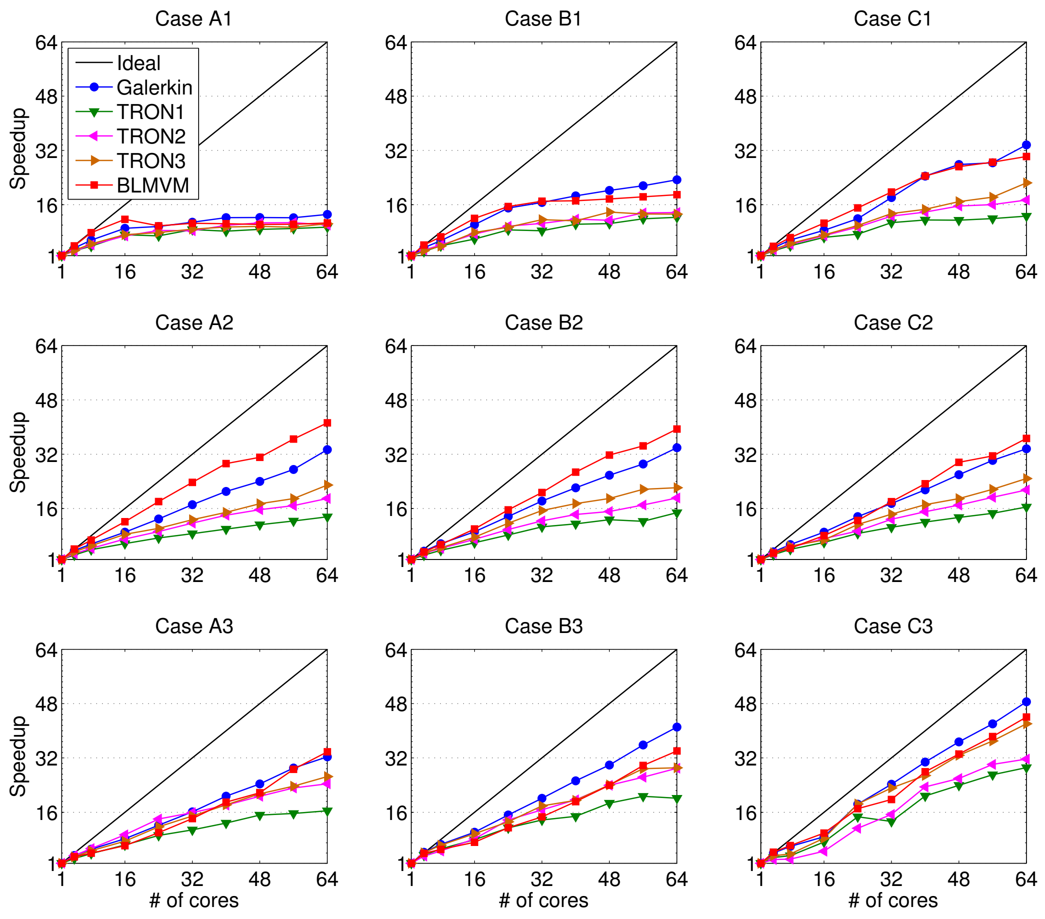


Figure 3.10: Cube with a hole: speedup for all 9 mesh cases up to 64 processors on the Mustang system (16 cores per node).

MPI process decreases. We can infer from these results that although BLMVM is the more efficient optimization in the hardware sense, TRON is more efficient in the algorithmic sense due to its lesser time-to-solution. Our study has shown that one can draw correlations between the performance models conducted on a single-core and the actual speedup across multiple nodes on the same network and switch. As future solvers and algorithms are implemented within PETSc, we can use this performance model to assess how efficient they are in both the hardware and algorithmic sense and how efficiently they will scale in a parallel setting.

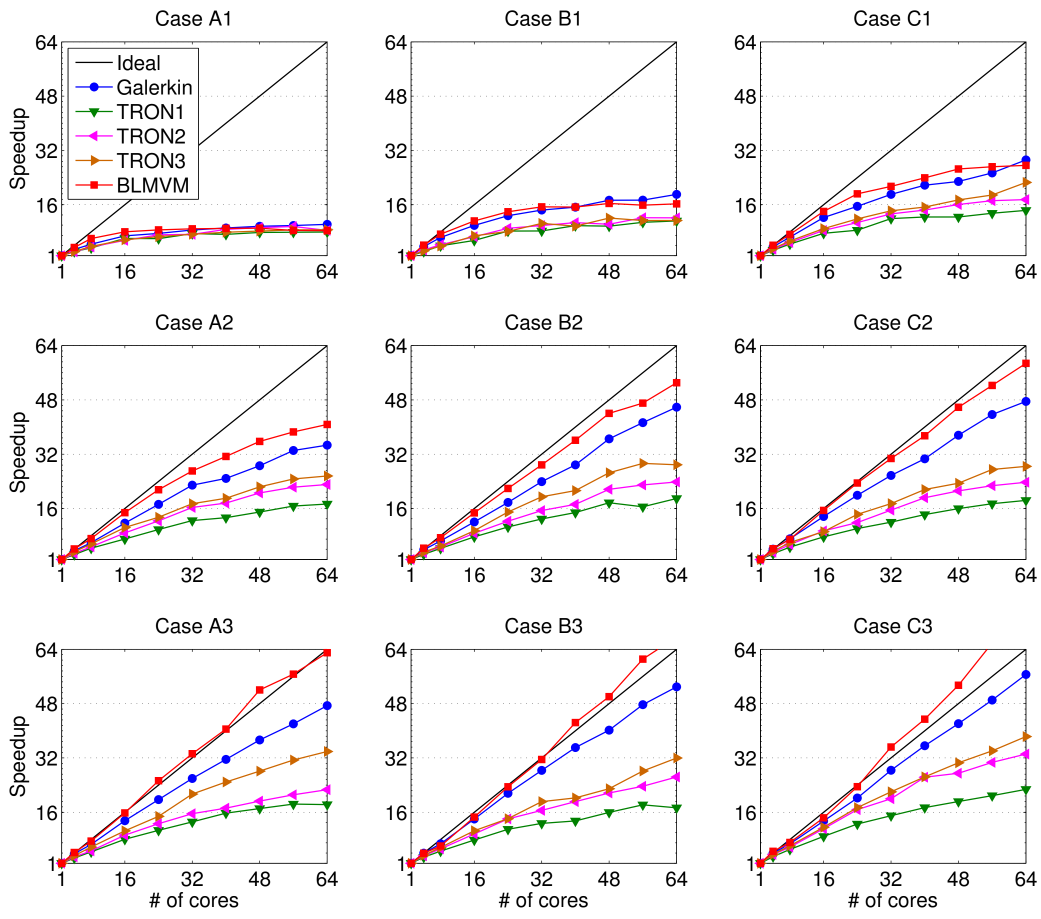


Figure 3.11: Cube with a hole: speedup for all 9 mesh cases up to 64 cores on the Wolf system (8 cores per node).

3.5.2 Transport of chromium in subsurface

Subsurface clean-up due to anthropogenic contamination is a big challenge [48]. Remediation studies [67, 70] need accurate predictions of transport of the involved chemical species, which are obtained using limited data at monitoring wells and through numerical simulations. To accurately predict the fate of the contaminant, a transport solver that: a) is robust, in that it will not give unphysical solutions, and b) can handle field-scale scenarios, is needed. The computational framework that is proposed in this chapter is an ideal candidate for such problems. We now consider

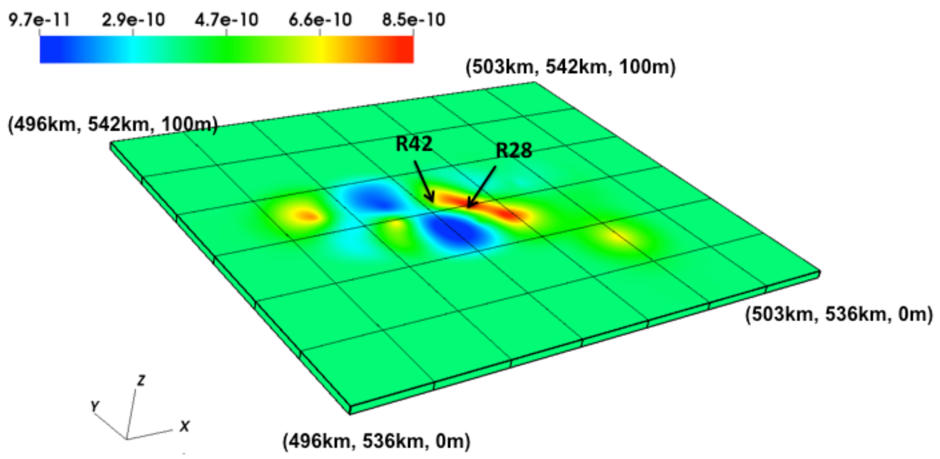


Figure 3.12: Chromium plume migration in the subsurface: Permeability field (m^2) and the locations of the pumping well (R28) and contaminant source (R42).

Table 3.7: Chromium plume migration in the subsurface: parameters

Parameter	Value
α_L	100 m
α_T	0.1 m
Contaminant source (R42)	1×10^{-4} kg/m^{2}s^2
Δt	0.2 days
Domain size	7000 km$\times$6000 km$\times$100 m
D_M	1×10^{-9} m^2/s
Permeability	Varies
Pumping well (R28)	-0.01 kg/m^{2}s^2
Total hexahedrons	1,984,512
Total vertices	2,487,765
$\mathbf{v}$	Varies with position
Viscosity	3.95$\times$10^{-5} Pa s

a realistic large-scale problem to predict the fate of chromium in the Los Alamos, New Mexico area. The chromium was released into the Sandia canyon in the 1950s up to early 1970s. Back then chromium was used as an anti-corrosion agent for the cooling towers at a power plant at the Los Alamos National Laboratory (see [71] and references therein for details).

Here we study the effectiveness of our proposed framework to this real-world scenario of predicting the extent of chromium plume. The following is a conceptual model domain that is considered: A domain of size $[496, 503]$km $\times$ $[536, 542]$km $\times$

$[0, 100]$m with the permeability field (m^2) as shown in Figure 3.12. R42 in Figure 3.12 is estimated to be the contaminant source location and a pumping well is located at R28. The parameters used for this problem are shown in Table 3.7, and we employ the following boundary conditions:

$$c^{\mathrm{p}}(x = 496\mathrm{km}, y, z) = 0, \tag{3.5.3a}$$

$$c^{\mathrm{p}}(x = 503\mathrm{km}, y, z) = 0, \tag{3.5.3b}$$

$$c^{\mathrm{p}}(x, y = 536\mathrm{km}, z) = 0, \quad \text{and} \tag{3.5.3c}$$

$$c^{\mathrm{p}}(x, y = 542\mathrm{km}, z) = 0. \tag{3.5.3d}$$

For this highly heterogeneous problem, we employ PETSc's algebraic multi-grid preconditioner (GAMG) and couple this with the TRON algorithm for the non-negative solver. Our goal is to examine its strong-scaling potential across 1024 cores. We first solve the steady flow equation (based on mass balance and Darcy's model to relate pressure and mass flux) with the pumping well located at R28. Cell-wise velocity is obtained from the resulting pressure field and used to calculated element-wise dispersion tensor. We then solve the transient diffusion problem (with tensorial dispersion) with a constant contaminant source located at R40 for up to 180 days. The concentrations at select time levels for Galerkin formulation and DMP methodology are shown in Figures 3.13 and 3.14, respectively. Negative concentrations arise with the Galerkin formulation even as the solution approaches steady-state.

Figure 3.15 depicts the amount of wall-clock time with respect to the number of cores at the first time step. The problem size is roughly 2.5 million degrees-of-freedom so if 1024 cores are utilized, each core only manages approximately 2,400 degrees-of-freedom, thus interprocess data exchange time will likely outweigh the computation time. However, we note that the parallel efficiency at the 64 core mark (4 and 8 node respectively) is roughly 88 percent, whereas the relative parallel efficiency from 16

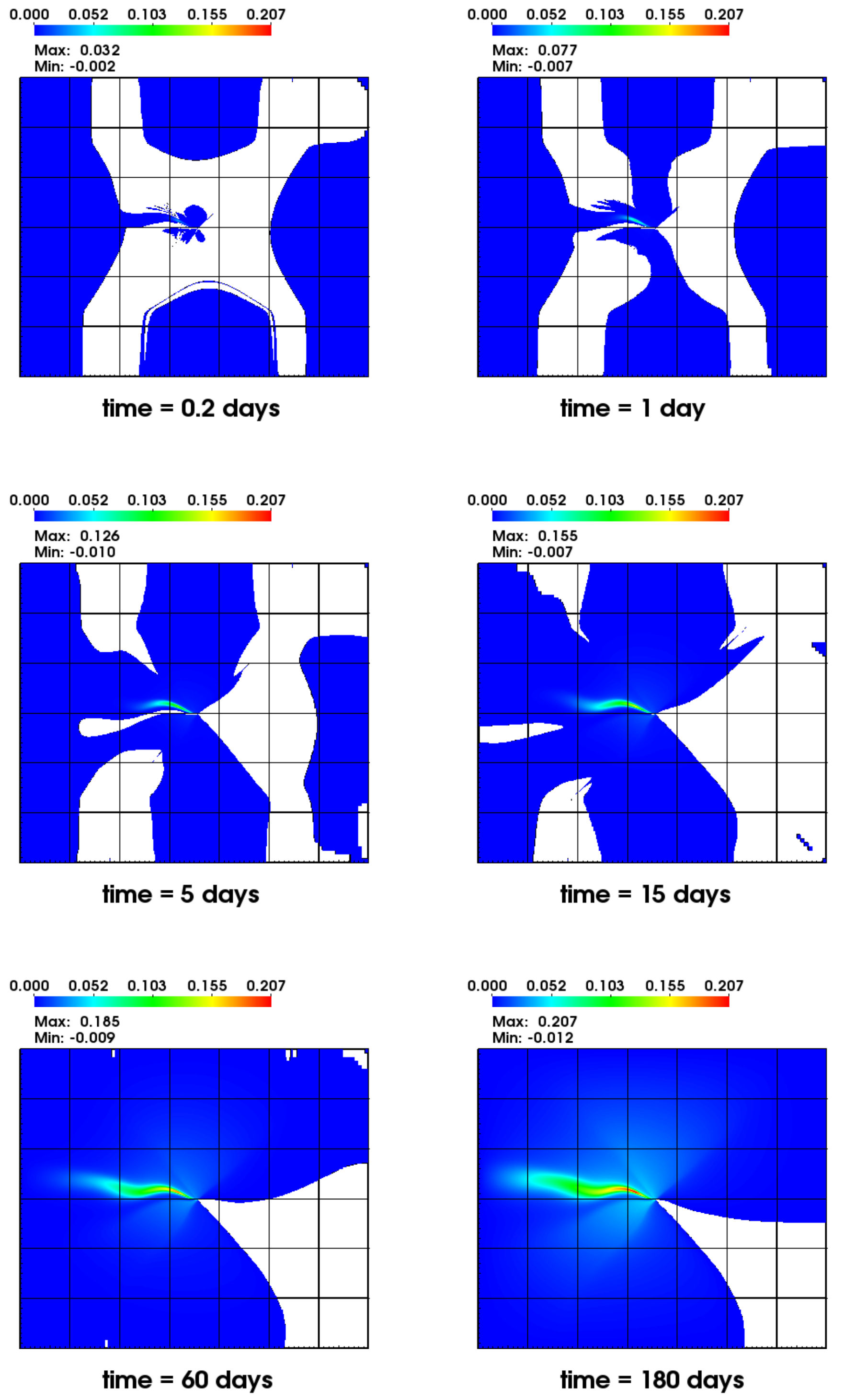


Figure 3.13: Chromium plume migration in the subsurface: concentrations at select times using the Galerkin formulation.

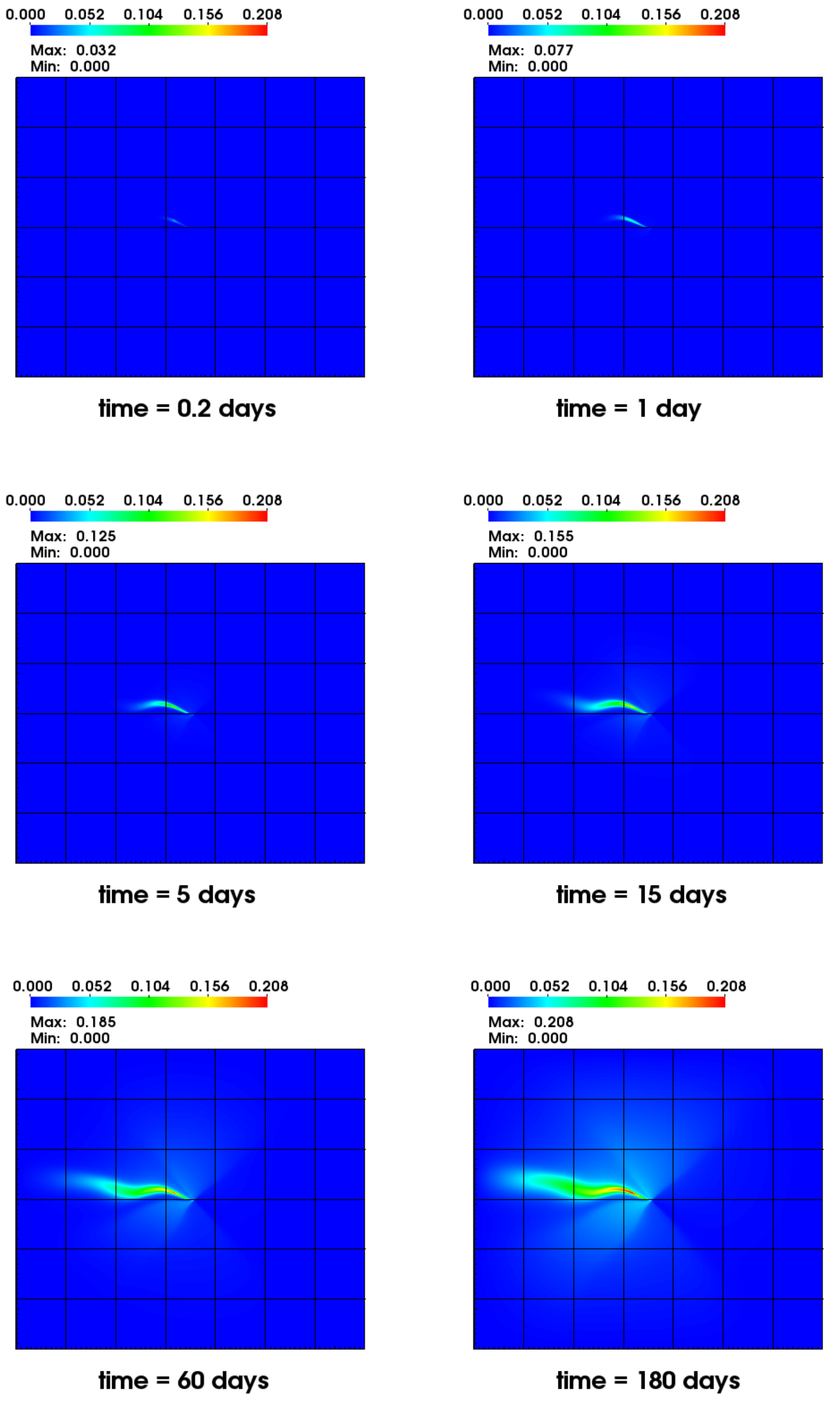


Figure 3.14: Chromium plume migration in the subsurface: concentrations at select times using the non-negative methodology.

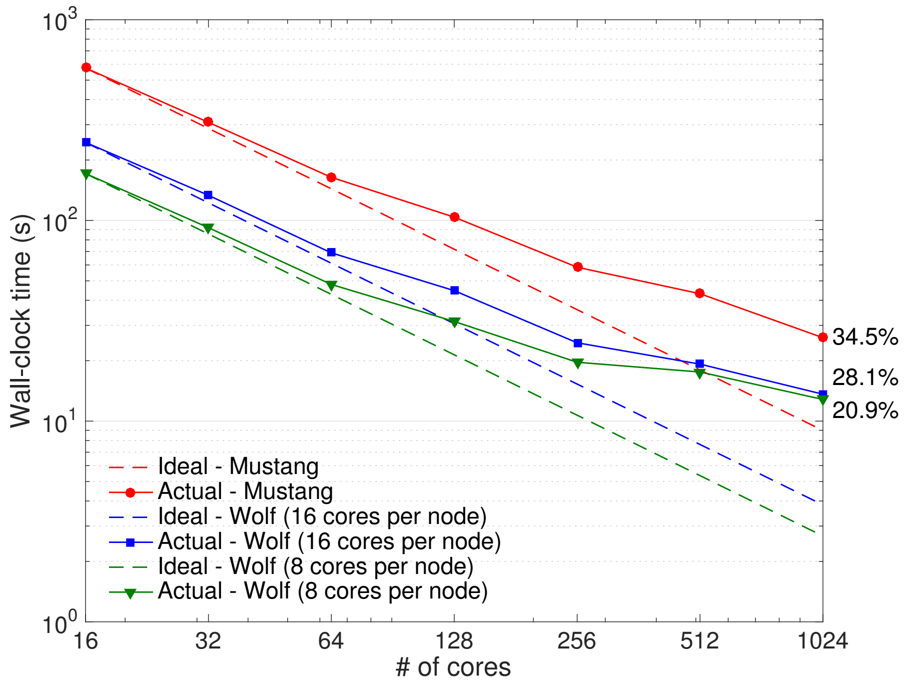


Figure 3.15: Chromium plume migration in the subsurface: wall-clock time of the TRON optimization solver with multi-grid preconditioner (GAMG) versus number of processors after the first time level.

to 64 cores for the unit cube A3 TRON3 case is about 80 percent parallel efficiency. These metrics suggest that TRON with GAMG demonstrates strong-scaling potential comparable to that of the TRON with ILU combination. A more thorough analysis of the potential bottlenecks presented by TRON and GAMG method can be done when a performance model for sparse matrix-matrix is developed.

It is also interesting to note that the wall-clock time is affected by whether or not a compute node is fully saturated. When a Wolf node utilizes all 16 cores per node, the performance with respect to wall-clock time is worse than when a Wolf node utilizes only 8 of 16 cores per node. The reason for this change is because the code becomes memory bandwidth limited; it decreases as more cores are utilized as shown from Figure 3.2 because of contention due to "over subscription". Therefore, it is recommended that to maximize the performance of these optimization-based solvers, the number of cores used per node is governed by the memory bandwidth saturation.

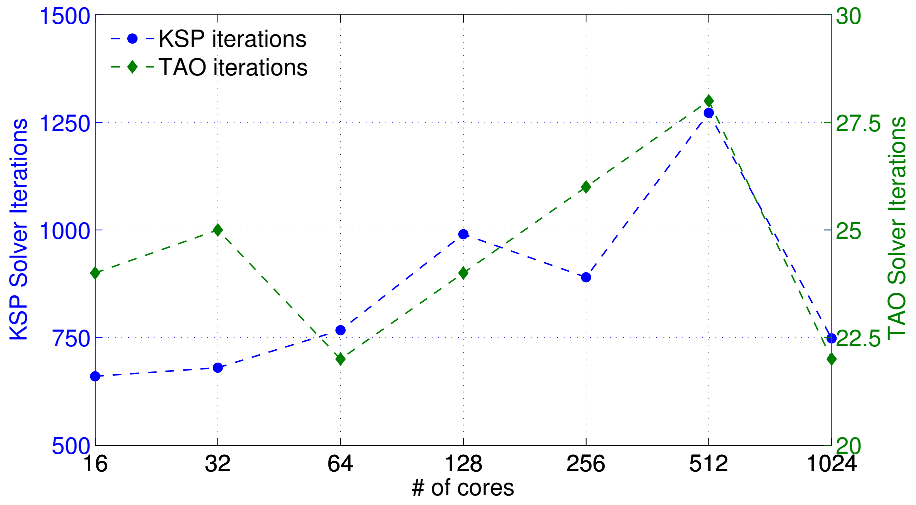


Figure 3.16: Chromium plume migration in the subsurface: number of KSP (left hand side) and TAO (right hand side) solver iterations for the TRON optimization solver versus number of cores after the first time level.

Another metric of interest is the number of solver iterations required for convergence across various numbers of MPI processes. Figure 3.16 depicts the number of KSP solver iterations and TAO solver iterations for 1024 cores, and we notice that there are significant fluctuations. This trend is largely attributed to the accumulation of numerical round-offs from the TRON algorithm. One can reduce these fluctuations by tightening the solver tolerances, but the strong-scaling remains largely unaffected even for the results shown. This study suggests that the proposed non-negative methodology using TRON with GAMG preconditioning is suitable for large-scale transient heterogeneous and anisotropic diffusion equations.

3.6 CONCLUDING REMARKS

We presented a parallel DMP computational framework suitable for solving large-scale steady-state and transient anisotropic diffusion equations. The main contribution is that the proposed parallel computational framework satisfies the DMP for large-scale diffusion-type equations even on general computational grids. The parallel framework is built upon PETSc's DMPlex data structure, which can handle unstructured meshes, and TAO for solving the resulting optimization problems from the discretization formulation. We have conducted systematic performance modeling and strong-scaling studies to demonstrate the efficiency, both in the parallel and hardware sense of the computational framework. The robustness of the proposed framework has been illustrated by solving a large-scale realistic problem involving the transport of chromium in the subsurface at Los Alamos, New Mexico. In the next chapter, we extend the proposed parallel framework to advective-diffusive using variational inequalities.

Chapter 4. Variational inequality approach to enforcing the non-negative constraint for advection-diffusion equations

4.1 INTRODUCTION

It has been shown in [30] that the PETSc/TAO based computational framework can successfully ensure non-negative concentrations in a large-scale setting. This chapter extends that work by presenting a numerical methodology based on *variational inequalities* for *anisotropic* diffusion and advection-diffusion equations that satisfies discrete maximum principles, meets the non-negative constraint, and is well-suited for solving large-scale problems using *parallel computing*.

The diffusion and advection-diffusion equations are important partial differential equations which are commonly used to model flow and transport of chemical species in porous media. Some of the applications include subsurface remediation [48, 70, 71] and transport of radionuclides [67, 56]. Since these important problems are not analytically tractable, one needs to rely on predictive numerical simulations. An important aspect in a predictive simulation of these equations is to satisfy the non-negative constraint of concentration of chemical species.

Research efforts over the years have successfully created numerical models and discretization for these equations, but they are not without their setbacks. For example, non-monotone discretizations like the finite element method may result in spurious oscillations with high Péclet numbers. Other common issues that may occur within highly heterogeneous and anisotropic diffusion-type equations are violations of the maximum principle and the non-negative constraint [37, 104, 105, 56]. Such numerical setbacks can result in algorithmic failures or sharp fronts that may result in erroneous approximations of reactive transport. Moreover, several important applications which require accurate predictive capabilities of transport solvers are often large-scale and cannot be solved on a single computer. It is important for numerical algorithms to not only ensure maximum principle but scale well with respect to both problem size and computing concurrency. Obtaining numerical solutions within a reasonable amount of time is the ultimate goal when selecting or designing algorithms that are robust and can ensure non-negative concentrations for a wide range of subsurface transport applications.

4.1.1 Our approach and its salient features

The main contribution of this chapter is to present a finite element computational framework applicable to both diffusion and advection-diffusion equations that meets the maximum principle and satisfies the non-negative constraint. The framework is built by rewriting the weak formulation (WF) as a variational inequality (VI) [160].

The field of VIs grew from a problem posed by Antonio Signorini [153, 154]. This problem was later coined as "Signorini problem" by Gaetano Fichera, who was a student of Signorini. Fichera posed the problem more precisely and obtained a variational inequality corresponding to the problem using which he established existence and uniqueness of solutions [52, 53]. VIs have been employed to study contact problems [83, 76], obstacle problems [145], elastoplastic problems [76, 68] and other problems arising in mechanics and mathematics [84]. If the bilinear form under the WF is symmetric, one can rewrite the WF as a quadratic programming (QP), which is a special case of VIs [36]. Most of the existing *single-field* WFs for advection-diffusion equations do not have symmetric bilinear forms, and hence one cannot construct equivalent problems under QP. To the best of the authors' knowledge, VIs have not be employed to develop numerical formulations to satisfy maximum principles and the non-negative constraint for anisotropic advection-diffusion equations.

The framework is particularly suited for large-scale problems, which is the case with many practical subsurface applications. The proposed framework enjoys the following salient features:

1. One can enforce bounded constraints for any transport problems that may be non-symmetric or nonlinear.

2. One can employ any numerical formulation, even a single-field formulation, for solving advection-diffusion equations.

3. One can leverage on existing high performance computing libraries and toolkits (e.g., solvers and preconditioners).

4. The framework is amenable for parallel computing, which will be illustrated using both strong and weak scaling studies.

The rest of the chapter is organized as follows. In Section 4.2, we present the boundary value problem for steady-state diffusion and advection-diffusion equations. In Section 4.3, we present the variational inequality (VI) and the various single-field weak formulations (WF) in the continuous setting. In Section 4.4, we propose the computational framework in a discrete setting and discuss in detail the specific solvers and implementation procedure. In Section 4.5, numerical results for the steady-state governing equations under the proposed framework are shown, and we conduct a thorough strong and weak-scaling study to demonstrate the parallel performance. In Section 4.6, we provide an extension of the proposed framework to transient problems and illustrate the performance of miscible displacement in porous media, which is a coupled non-linear phenomenon. Conclusions are drawn in Section 4.7. To facilitate the reader to be able to reproduce the results given in this chapter, sample Firedrake project files along with the discussion on the solution strategy for large-scale Darcy equations are provided in Appendices A and B.

4.2 GOVERNING EQUATIONS IN THE CONTINUOUS SETTING

Let $\Omega \subset \mathbb{R}^d$ be a bounded open domain, where 'd' denotes the number of spatial dimensions. The boundary is denoted by $\partial\Omega = \overline{\Omega} - \Omega$, where a superposed bar denotes the set closure. We denote the set of all k-times continuously differentiable functions on Ω by $C^k(\Omega)$. We denote the set of all functions in $C^0(\Omega)$ that are continuous to the boundary by $C^0(\overline{\Omega})$. A spatial point is denoted by $\mathbf{x} \in \overline{\Omega}$. The gradient and divergence operators with respect to $\mathbf{x}$ are, respectively, denoted by grad$[\cdot]$ and div$[\cdot]$. The unit outward normal to boundary is denoted by $\widehat{\mathbf{n}}(\mathbf{x})$.

Let $c(\mathbf{x})$ denote the concentration field. The boundary is divided into two parts: Γ^{D} and Γ^{N}, such that $\Gamma^{\mathrm{D}} \cup \Gamma^{\mathrm{N}} = \partial\Omega$ and $\Gamma^{\mathrm{D}} \cap \Gamma^{\mathrm{N}} = \emptyset$. Γ^{D} is that part of the boundary on which Dirichlet boundary conditions are enforced (i.e., concentration is prescribed). Γ^{N} is the part of the boundary on which Neumann boundary conditions are enforced (i.e., flux is prescribed). When advection is present, the Neumann boundary is further divided into inflow and outflow regions, which are defined as follows:

$$\Gamma^{\mathrm{N}}_{\mathrm{inflow}} := \left\{ \mathbf{x} \in \Gamma^{\mathrm{N}} \;\middle|\; \mathbf{v}(\mathbf{x}) \cdot \widehat{\mathbf{n}}(\mathbf{x}) < 0 \right\} \quad \text{and} \tag{4.2.1a}$$

$$\Gamma^{\mathrm{N}}_{\mathrm{outflow}} := \left\{ \mathbf{x} \in \Gamma^{\mathrm{N}} \;\middle|\; \mathbf{v}(\mathbf{x}) \cdot \widehat{\mathbf{n}}(\mathbf{x}) \geq 0 \right\}. \tag{4.2.1b}$$

For uniqueness of the solution under a steady-state response, we assume that concentration is prescribed on a non-zero part of the boundary (i.e., meas$\left(\Gamma^{\mathrm{D}}\right) > 0$).

4.2.1 Strong problems (SP)

The strong problem (SP) for steady-state diffusion reads: Find $c(\mathbf{x}) \in C^2(\Omega) \cap C^0(\overline{\Omega})$ such that we have

$$
\begin{aligned}
-\mathrm{div}[\mathbf{D}(\mathbf{x})\mathrm{grad}[c(\mathbf{x})]] &= f(\mathbf{x}) && \text{in } \Omega, && (4.2.2a)\\
c(\mathbf{x}) &= c^{\mathrm{p}}(\mathbf{x}) && \text{on } \Gamma^{\mathrm{D}}, \quad \text{and} && (4.2.2b)\\
-\widehat{\mathbf{n}}(\mathbf{x}) \cdot \mathbf{D}(\mathbf{x})\mathrm{grad}[c(\mathbf{x})] &= q^{\mathrm{p}}(\mathbf{x}) && \text{on } \Gamma^{\mathrm{N}}, && (4.2.2c)
\end{aligned}
$$

and the SP for steady-state advection-diffusion reads: Find $c(\mathbf{x}) \in C^2(\Omega) \cap C^0(\overline{\Omega})$ such that we have

$$
\begin{aligned}
&\mathbf{v}(\mathbf{x}) \cdot \mathrm{grad}[c(\mathbf{x})] - \mathrm{div}[\mathbf{D}(\mathbf{x})\mathrm{grad}[c(\mathbf{x})]] = f(\mathbf{x}) && \text{in } \Omega, && (4.2.3a)\\
&c(\mathbf{x}) = c^{\mathrm{p}}(\mathbf{x}) && \text{on } \Gamma^{\mathrm{D}}, && (4.2.3b)\\
&\widehat{\mathbf{n}}(\mathbf{x}) \cdot (\mathbf{v}(\mathbf{x})c(\mathbf{x}) - \mathbf{D}(\mathbf{x})\mathrm{grad}[c(\mathbf{x})]) = q^{\mathrm{p}}(\mathbf{x}) && \text{on } \Gamma^{\mathrm{N}}_{\mathrm{inflow}}, \quad \text{and} && (4.2.3c)\\
&-\widehat{\mathbf{n}}(\mathbf{x}) \cdot \mathbf{D}(\mathbf{x})\mathrm{grad}[c(\mathbf{x})] = q^{\mathrm{p}}(\mathbf{x}) && \text{on } \Gamma^{\mathrm{N}}_{\mathrm{outflow}}, && (4.2.3d)
\end{aligned}
$$

where $\mathbf{v}(\mathbf{x})$ is the advective velocity, $f(\mathbf{x})$ is the prescribed volumetric source/sink, $c^{\mathrm{p}}(\mathbf{x})$ is the prescribed concentration on the boundary, $q^{\mathrm{p}}(\mathbf{x})$ is the prescribed flux on the boundary, and $\mathbf{D}(\mathbf{x})$ is the second-order diffusivity tensor. The diffusivity tensor is assumed to be bounded and uniformly elliptic. That is, there exist two constants $0 < \xi_1 \leq \xi_2 < +\infty$ such that

$$\xi_1 \mathbf{y} \cdot \mathbf{y} \leq \mathbf{y} \cdot \mathbf{D}(\mathbf{x})\mathbf{y} \leq \xi_2 \mathbf{y} \cdot \mathbf{y} \quad \forall \mathbf{y} \in \mathbb{R}^d. \tag{4.2.4}$$

Moreover, the diffusivity tensor is assumed to be symmetric. That is,

$$\mathbf{D}(\mathbf{x}) = \mathbf{D}^{\mathrm{T}}(\mathbf{x}) \quad \forall \mathbf{x} \in \Omega. \tag{4.2.5}$$

A solution to SP is commonly referred to as a *classical* solution.

4.2.2 Maximum principle and the non-negative constraint

From the theory of partial differential equations, it is well-known that a classical solution for the above mentioned SPs satisfies maximum principles. For completeness, we provide below the statement of the classical maximum principle of second-order elliptic partial differential equations with Dirichlet boundary conditions on the entire boundary.

Theorem 4.2.1. *(Classical maximum principle) If* $\Gamma^{\mathrm{D}} = \partial\Omega$*,* $c(\mathbf{x}) \in C^2(\Omega) \cap C^0(\overline{\Omega})$ *and* $f(\mathbf{x}) \leq 0$*, then*

$$\max_{\mathbf{x}\in\Omega} \; c(\mathbf{x}) \leq \max_{\mathbf{x}\in\partial\Omega} \; c^{\mathrm{p}}(\mathbf{x}). \tag{4.2.6}$$

Proof. A proof can be found in [60]. □

A generalization of the classical maximum principle that is relevant to this chapter is provided in [120]. Specifically, they have extended the classical maximum principle on four fronts: the regularity of the solution is relaxed to $C^1(\Omega) \cap C^0(\overline{\Omega})$, the regularity of the volumetric source $f(\mathbf{x})$ is relaxed to the space of square integrable functions, the boundary can have both Dirichlet and Neumann boundary conditions, and the Neumann boundary conditions are further divided into inflow and outflow (i.e., similar to equations (4.2.1a)–(4.2.1b)). For the sake of brevity, we defer all interested readers to the suggested reference. Another property that is relevant to this chapter is non-negative solutions, which can be shown to be a special case of maximum principles. In particular, the above maximum principle implies the following result:

Corollary 4.2.2. *(Non-negative solutions) If* $\Gamma^{\mathrm{D}} = \partial\Omega$*,* $c(\mathbf{x}) \in C^2(\Omega) \cap C^0(\overline{\Omega})$*,*

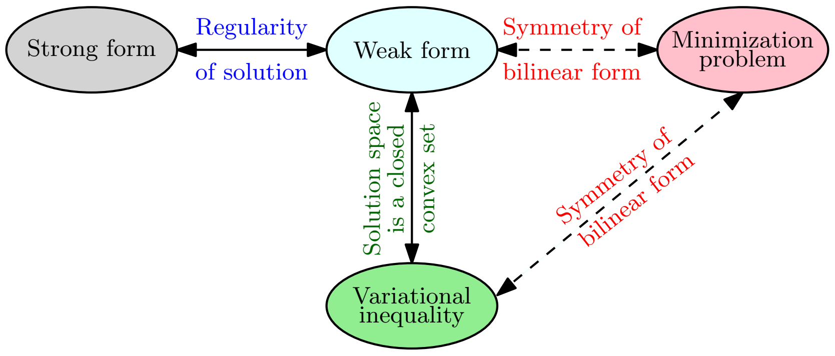


Figure 4.1: Relationships between the strong problem (SP), a weak formulation (WF), a variational inequality (VI), and the minimization problem (MP).

$c^{\mathrm{p}}(\mathbf{x}) \geq 0$, *and* $f(\mathbf{x}) \geq 0$, *then*

$$0 \leq c(\mathbf{x}) \quad \forall \mathbf{x} \in \overline{\Omega}. \tag{4.2.7}$$

The central aim of this chapter is to obtain numerical solutions to the above governing equations (i.e., equations (4.2.2) *and* (4.2.3)*) that respect maximum principles and the non-negative constraint.*

The main task will then be to find an appropriate setting for numerical solutions. The finite difference method directly discretizes the SP. However, under the finite element method, the SP is rewritten as a WF, which is equivalent to the SP under some regularity assumptions. A solution to a WF is referred to as a weak solution. As mentioned in Section 4.1 and will be shown using several examples later in this chapter, a WF does not guarantee non-negative solutions in the discrete setting. To overcome this deficiency, some non-negative formulations, especially for diffusion-type equations, have rewritten the WF as an equivalent minimization problem (MP) and augmented with bound constraints. However, it needs to be emphasized that such conversion is not always possible, which is the case with the typical WF for advection-diffusion equations, as these formulations have non-symmetric bilinear forms. In order to handle non-self-adjoint differential operators (e.g., advection-diffusion equation) and WFs with non-symmetric bilinear forms, we rewrite a given WF as a VI. In order to satisfy maximum principles and the non-negative constraint, we restrict the feasible solution space of the VI formulation using bound constraints. It needs to be mentioned that one can pose the VI as an equivalent MP only if the bilinear form is symmetric. Figure 4.1 illustrates the various ways of rewriting the governing equations, and the conditions under which one form is equivalent to the other. We now present various WFs for diffusion and advection-diffusion equations, which will form the basis for our proposed VI-based formulations.

4.3 VARIATIONAL INEQUALITIES AND WEAK FORMULATIONS

The non-negative constraint and maximum principles restrict the feasible solution space to a closed convex set. A variational inequality (VI) is basically a variational problem on a convex set, which need not be a vector space. To this end, let $\mathcal{C}$ denote the solution space for the concentration field, and $\mathcal{K}$ be a closed convex subset of $\mathcal{C}$. The subset $\mathcal{K}$ is defined by the underlying maximum principles and the non-negative constraint. The formulation based on VIs corresponding to the mentioned SPs can be compactly written as: Find $c(\mathbf{x}) \in \mathcal{K}$ such that we have

$$\mathcal{B}(w - c; c) \geq \mathcal{L}(w - c) \quad \forall w(\mathbf{x}) \in \mathcal{K}, \tag{4.3.1}$$

where $\mathcal{B}(\cdot;\cdot)$ is a bilinear form and $\mathcal{L}(\cdot)$ is a linear functional, whose specific choices are provided by the associated weak formulation. A WF can be abstractly written as: Find $c(\mathbf{x}) \in \mathcal{C}$ such that we have

$$\mathcal{B}(w; c) = \mathcal{L}(w) \quad \forall w(\mathbf{x}) \in \mathcal{W}, \tag{4.3.2}$$

where $\mathcal{C}$ and $\mathcal{W}$ are appropriate function spaces for a given WF. Our intention is to illustrate the applicability of the proposed VI framework to a variety of WFs. To this end, we employ the continuous Galerkin (GAL), Streamlined Upwind Petrov-Galerkin

(SUPG), and Discontinuous Galerkin (DG) formulations, which are documented below. For convenience, the standard L_2 inner-product over K is denoted as follows:

$$(a; b)_K = \int_K a(\mathbf{x}) \cdot b(\mathbf{x}) \, \mathrm{d}K. \tag{4.3.3}$$

4.3.1 Continuous Galerkin

The relevant function spaces are:

$$\mathcal{C} := \left\{ c(\mathbf{x}) \in H^1(\Omega) \;\middle|\; c(\mathbf{x}) = c^{\mathrm{p}}(\mathbf{x}) \text{ on } \Gamma^{\mathrm{D}} \right\} \quad \text{and} \tag{4.3.4}$$

$$\mathcal{W} := \left\{ w(\mathbf{x}) \in H^1(\Omega) \;\middle|\; w(\mathbf{x}) = 0 \text{ on } \Gamma^{\mathrm{D}} \right\}, \tag{4.3.5}$$

where $H^1(\Omega)$ is a Sobolev space [22]. We assume that $f(\mathbf{x}) \in H^{-1}(\Omega)$, which is a dual space corresponding to $H^1(\Omega)$. We employ the GAL formulation for the diffusion problem, for which the bilinear form and linear functional are:

$$\mathcal{B}_{\mathrm{GAL}}(w; c) := \Big(\mathrm{grad}[w(\mathbf{x})];\ \mathbf{D}(\mathbf{x})\mathrm{grad}[c(\mathbf{x})]\Big)_{\Omega} \quad \text{and} \tag{4.3.6}$$

$$\mathcal{L}_{\mathrm{GAL}}(w) := \Big(w(\mathbf{x});\ f(\mathbf{x})\Big)_{\Omega} - \Big(w(\mathbf{x});\ q^{\mathrm{p}}(\mathbf{x})\Big)_{\Gamma^{\mathrm{N}}}. \tag{4.3.7}$$

For the advection-diffusion problem, spurious oscillations may arise under the GAL formulation for high Péclet numbers. Herein, we employ the SUPG formulation [23], and the corresponding bilinear form and linear functional are:

$$\mathcal{B}_{\mathrm{SUPG}}(w; c) := \mathcal{B}_{\mathrm{RES}}(w; c) + \Big(w(\mathbf{x});\ \mathbf{v}(\mathbf{x}) \cdot \mathrm{grad}[c(\mathbf{x})]\Big)_{\Omega} + \Big(\mathrm{grad}[w(\mathbf{x})];\ \mathbf{D}(\mathbf{x})\mathrm{grad}[c(\mathbf{x})]\Big)_{\Omega} \quad \text{and} \tag{4.3.8}$$

$$\mathcal{L}_{\mathrm{SUPG}}(w) := \mathcal{L}_{\mathrm{RES}}(w) + \Big(w(\mathbf{x});\ f(\mathbf{x})\Big)_{\Omega} - \Big(w(\mathbf{x});\ q^{\mathrm{p}}(\mathbf{x})\Big)_{\Gamma^{\mathrm{N}}}, \tag{4.3.9}$$

where the residual terms

$$\mathcal{B}_{\mathrm{RES}}(w;c) := \left(\frac{h}{2\|\mathbf{v}(\mathrm{x})\|}\mathbf{v}(\mathrm{x}) \cdot \mathrm{grad}[w(\mathbf{x})];\ \mathbf{v}(\mathrm{x}) \cdot \mathrm{grad}[c(\mathbf{x})] - \mathrm{div}\left[\mathbf{D}(\mathbf{x})\mathrm{grad}[c(\mathbf{x})]\right]\right)_{\Omega} \quad \text{and} \tag{4.3.10}$$

$$\mathcal{L}_{\mathrm{RES}}(w) := \left(\frac{h}{2\|\mathbf{v}(\mathrm{x})\|}\mathbf{v}(\mathrm{x}) \cdot \mathrm{grad}[w(\mathbf{x})];\ f(\mathrm{x})\right)_{\Omega}, \tag{4.3.11}$$

and h denotes the element-wise diameter.

4.3.2 Discontinuous Galerkin

For several transport applications, it is highly desirable to possess element-wise mass balance property, as it is an important fundamental physical law [159]. This is particularly true when the transport is coupled with chemical reactions and biofilm growth [163, 161]. The GAL and SUPG formulations do not possess this property without any further modification or enrichment to their formulations. One way to ensure this property under a single-field finite element framework is through the use of the DG formulations (see [9, 144, 38, 96, 95, 97, 133] and the references within for further details). To present the DG formulation employed in this chapter, we now introduce relevant notation.

The domain Ω is divided into S subdomains

$$\Omega = \bigcup_{i=1}^{S} \omega_i. \tag{4.3.12}$$

The boundary of the subdomain ω_i is denoted by $\partial\omega_i$. The interior face between ω_i and ω_j is denoted by Γ_{ij}. That is,

$$\Gamma_{ij} = \partial\omega_i \cap \partial\omega_j. \tag{4.3.13}$$

The set of all points on the interior faces is denoted by Γ_{int}. Mathematically,

$$\Gamma_{\mathrm{int}} = \bigcup_{i=1, i<j}^{S} \Gamma_{ij}. \tag{4.3.14}$$

For an interior face, we denote the subdomains shared by this face by ω^+ and ω^-. The outward normals on this face for these subdomains are, respectively, denoted by $\widehat{\mathbf{n}}^+$ and $\widehat{\mathbf{n}}^-$. Employing Brezzi's notation [9], the average and jump operators on an interior face are defined as follows

$$\Big\{c\Big\} := \frac{c^+ + c^-}{2} \quad \text{and} \quad \Big[\!\!\Big[c\Big]\!\!\Big] := c^+ \widehat{\mathbf{n}}^+ + c^- \widehat{\mathbf{n}}^-, \tag{4.3.15}$$

where

$$c^+ = c|_{\partial\omega^+} \quad \text{and} \quad c^- = c|_{\partial\omega^-}. \tag{4.3.16}$$

One of the most popular DG formulations is the *Interior Penalty* method, which for equation (4.2.2) is written as

$$\begin{aligned}
\mathcal{B}_{\mathrm{DG}}(w; c) := &\Big(\mathrm{grad}[w(\mathbf{x})];\ \mathbf{D}(\mathbf{x})\mathrm{grad}[c(\mathbf{x})]\Big)_{\Omega} - \Big(\Big[\!\!\Big[w(\mathbf{x})\Big]\!\!\Big];\ \Big\{\mathbf{D}(\mathbf{x})\mathrm{grad}[c(\mathbf{x})]\Big\}\Big)_{\Gamma_{\mathrm{int}}} \\
&+ \epsilon\Big(\Big\{\mathbf{D}(\mathbf{x})\mathrm{grad}[w(\mathbf{x})]\Big\};\ \Big[\!\!\Big[c(\mathbf{x})\Big]\!\!\Big]\Big)_{\Gamma_{\mathrm{int}}} \\
&+ \frac{\gamma}{h}\Big(\Big[\!\!\Big[w(\mathbf{x})\Big]\!\!\Big];\ \Big[\!\!\Big[c(\mathbf{x})\Big]\!\!\Big]\Big)_{\Gamma_{\mathrm{int}}} \quad \text{and} \qquad (4.3.17) \\
\mathcal{L}_{\mathrm{DG}}(w) := &\Big(w(\mathbf{x});\ f(\mathbf{x})\Big)_{\Omega} - \Big(w(\mathbf{x});\ q^{\mathrm{p}}(\mathbf{x})\Big)_{\Gamma^{\mathrm{N}}}, \qquad (4.3.18)
\end{aligned}$$

where the penalty term $\gamma = 2\frac{(d+1)}{d}$ [151] for first-order elements and $\epsilon \in \{-1, 0, 1\}$ denotes the Symmetric, Incomplete, and Non-symmetric Interior Penalty methods respectively. For equation (4.2.3), the DG formulation can be written as

$$\begin{aligned}\mathcal{B}_{\mathrm{DG}}(w;c) := &\Big(\mathrm{grad}[w(\mathbf{x})];\ \mathbf{D}(\mathbf{x})\mathrm{grad}[c(\mathbf{x})]\Big)_{\Omega} - \Big(\big[\!\big[w(\mathbf{x})\big]\!\big];\ \big\{\mathbf{D}(\mathbf{x})\mathrm{grad}[c(\mathbf{x})]\big\}\Big)_{\Gamma_{\mathrm{int}}} \\ &+ \epsilon\Big(\big\{\mathbf{D}(\mathbf{x})\mathrm{grad}[w(\mathbf{x})]\big\};\ \big[\!\big[c(\mathbf{x})\big]\!\big]\Big)_{\Gamma_{\mathrm{int}}} + \frac{\gamma}{h}\Big(\big[\!\big[w(\mathbf{x})\big]\!\big];\ \big[\!\big[c(\mathbf{x})\big]\!\big]\Big)_{\Gamma_{\mathrm{int}}} \\ &- \Big(w(\mathbf{x});\ \mathbf{v}(\mathbf{x})\cdot\mathrm{grad}[c(\mathbf{x})]\Big)_{\Omega} - \Big(\big[\!\big[w(\mathbf{x})\big]\!\big];\ c^{\mathrm{up}}(\mathbf{x})\mathbf{v}(\mathbf{x})\Big)_{\Gamma_{\mathrm{int}}} \quad \text{and} \quad (4.3.19)\\ \mathcal{L}_{\mathrm{DG}}(w) := &\Big(w(\mathbf{x});\ f(\mathbf{x})\Big)_{\Omega} - \Big(w(\mathbf{x});\ q^{\mathrm{p}}(\mathbf{x})\Big)_{\Gamma^{\mathrm{N}}}, \qquad (4.3.20)\end{aligned}$$

where the upwinding term $c^{\mathrm{up}}(\mathbf{x})$ is defined as

$$c^{\mathrm{up}}(\mathbf{x}) = \begin{cases} c^{+}(\mathbf{x}) & \text{if} \quad \mathbf{v}(\mathbf{x})\cdot\widehat{\mathbf{n}}^{+}(\mathbf{x}) > 0, \\ c^{-}(\mathbf{x}) & \text{otherwise.}\end{cases} \tag{4.3.21}$$

For the remainder of this chapter, we shall consider only the Symmetric Interior Penalty method where $\epsilon = -1$.

4.3.3 A theoretical discussion

The bilinear form is assumed to be continuous (i.e., bounded above). That is, there exists a constant $\kappa_1 > 0$ such that

$$\mathcal{B}(w;c) \leq \kappa_1 \|c\| \|w\| \quad \forall c(\mathbf{x}), w(\mathbf{x}) \in \mathcal{C}. \tag{4.3.22}$$

In addition, the bilinear form is assumed to be coercive. That is, there exists a constant $\kappa_2 > 0$ such that

$$\kappa_2 \|c\|^2 \leq \mathcal{B}(c;c) \quad \forall c(\mathbf{x}) \in \mathcal{C}. \tag{4.3.23}$$

Recall that $\mathcal{L}(\cdot)$ is assumed to be a linear continuous functional on $\mathcal{C}$. Then, from the Lax-Milgram theorem [21], it is known that a unique solution exists under the WF.

Under the same conditions on the bilinear form and the linear functional, a unique solution exists for the associated VI if $\mathcal{K} \subset \mathcal{C}$ is a closed convex subset [103]. A solution of the VI is a solution of the WF if $\mathcal{C} = \mathcal{K}$. Moreover, if the bilinear form is symmetric, that is,

$$\mathcal{B}(w; c) = B(c; w) \tag{4.3.24}$$

then the WF and the VI are equivalent to the following MP: Find $c(\mathbf{x}) \in \mathcal{C}$ such that

$$\underset{c(\mathbf{x}) \in \mathcal{C}}{\text{minimize}} \quad \frac{1}{2}\mathcal{B}(c; c) - \mathcal{L}(c). \tag{4.3.25}$$

These relations are pictorially described in Figure 4.1. From a theoretical point of view, it is important to note that the VIs that we will be dealing with for steady-state problems will be elliptic of first kind. For further details on infinite-dimensional VIs, see [61, 46].

4.4 PROPOSED COMPUTATIONAL FRAMEWORK IN A DISCRETE SETTING

We denote the total number of degrees-of-freedom by "$ndofs$". We also denote the vector of ones by $\mathbf{1}$, as in Chapter 3. The component-wise inequalities are denoted by $\preceq$ and $\succeq$. That is,

$$\boldsymbol{a} \preceq \boldsymbol{b} \quad \text{implies that} \quad a_i \leq b_i \; \forall i \quad \text{and} \tag{4.4.1a}$$

$$\boldsymbol{a} \succeq \boldsymbol{b} \quad \text{implies that} \quad a_i \geq b_i \; \forall i. \tag{4.4.1b}$$

The vector of unknown nodal concentrations is denoted by $\boldsymbol{c}$, and the corresponding nodal source vector is denoted by $\boldsymbol{f}$. The coefficient matrix after a finite element discretization is denoted by $\boldsymbol{K}$. Note that the vectors $\boldsymbol{c}$ and $\boldsymbol{f}$ are of size $ndofs \times 1$, and the matrix $\boldsymbol{K}$ is of size $ndofs \times ndofs$. We denote the standard inner-product in Euclidean spaces by $\langle \cdot ; \cdot \rangle$. That is,

$$\langle \boldsymbol{a} ; \boldsymbol{b} \rangle = \sum_{i}^{ndofs} a_i b_i \quad \forall \boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^{ndofs}. \tag{4.4.2}$$

The formulation in the discrete setting will be posed as a mixed complementarity problem (MCP) [84]. For convenience, we define $\boldsymbol{h} \in \mathbb{R}^{ndofs}$ as

$$\boldsymbol{h} := \boldsymbol{K}\boldsymbol{c} - \boldsymbol{f}. \tag{4.4.3}$$

The corresponding MCP reads: Find $c_{\min}\mathbf{1} \preceq \boldsymbol{c} \preceq c_{\max}\mathbf{1}$ such that for each $i \in \{1, ..., ndofs\}$

$$h_i(\boldsymbol{c}) \geq 0 \quad \text{if } c_{\min} = c_i, \tag{4.4.4a}$$

$$h_i(\boldsymbol{c}) = 0 \quad \text{if } c_{\min} \leq c_i \leq c_{\max}, \quad \text{and} \tag{4.4.4b}$$

$$h_i(\boldsymbol{c}) \leq 0 \quad \text{if } c_i = c_{\max}, \tag{4.4.4c}$$

where $c_{\min}$ and $c_{\max}$, respectively, denote the minimum and maximum concentrations, which are provided by the maximum principle or the non-negative constraint. Simple complementarity conditions arise from the first-order optimality conditions in optimization. For bound-constrained optimization, $\boldsymbol{h}$ corresponds to the gradient of the objective functional. If one has only the non-negative constraints (i.e., $c_{\min} = 0$ and $c_{\max} = +\infty$), then the problem reduces to a non-linear complementarity problem, which is a special case of MCP. For details on non-linear complementarity problems, see [50]. Note that the feasible region, which is restricted by the bound constraints, form a parallelepiped, which is a convex set [18].

Let the feasible region $\mathcal{K}$ be a convex subset of $\mathbb{R}^{ndofs}$. In our case, the feasible region is restricted by constraints which are in the form of finite number of linear

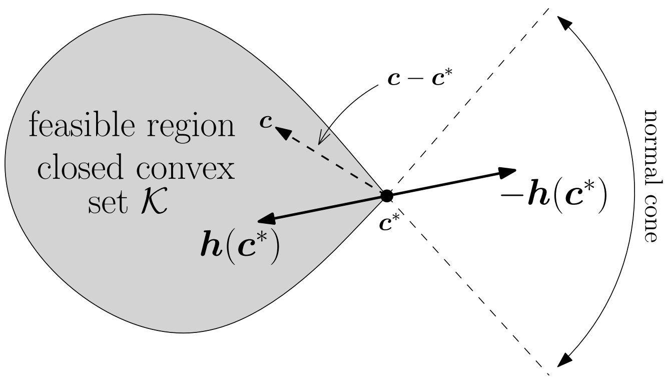

feasible region
closed convex
set $\mathcal{K}$
$\boldsymbol{c}-\boldsymbol{c}^*$
$\boldsymbol{c}$
$\boldsymbol{c}^*$
$\boldsymbol{h}(\boldsymbol{c}^*)$
$-\boldsymbol{h}(\boldsymbol{c}^*)$
normal cone

Figure 4.2: Condition under which a solution exists for a VI of the form $\langle \boldsymbol{h}(\boldsymbol{c}), \tilde{\boldsymbol{c}} - \boldsymbol{c}\rangle \geq 0 \; \forall \tilde{\boldsymbol{c}} \in \mathcal{K}$. Here, $\boldsymbol{c}^*$ denotes a solution of the VI. The normal cone of $\mathcal{K}$ at $\boldsymbol{c}^*$ is defined as $\mathcal{N}(\boldsymbol{c}^*) := \left\{\boldsymbol{w} \in \mathbb{R}^{ndofs} \;\middle|\; \langle \boldsymbol{w};\; \boldsymbol{c} - \boldsymbol{c}^*\rangle \leq \boldsymbol{0} \; \forall \boldsymbol{c} \in \mathcal{K}\right\}$.

equalities and inequalities. This makes the feasible region to be a polyhedron, which is a convex set [18]. It should be noted that bound constraints are a special case of linear inequalities. With this machinery at our disposal, one can pose the second formulation based on variational inequalities, which reads: Find $\boldsymbol{c} \in \mathcal{K}$ such that we have

$$\langle \boldsymbol{K}\boldsymbol{c}; \boldsymbol{v} - \boldsymbol{c}\rangle \geq \langle \boldsymbol{f}; \boldsymbol{v} - \boldsymbol{c}\rangle \quad \forall \boldsymbol{v} \in \mathcal{K}. \tag{4.4.5}$$

Note that MCP is a special case of VIs in which the feasible region is a parallelepiped (i.e., one has only bound constraints). The conditions under which a solution exists for the finite-dimensional VI given in equation (4.4.5) is pictorially described in Figure 4.2.

If the coefficient matrix $\boldsymbol{K}$ is symmetric, one can alternatively enforce maximum principles and the non-negative constraint using QP, which has been illustrated in [127, 131] for small-scale problems, and in [30] for large-scale problems in parallel environments. Therefore, this approach is only applicable for formally self-adjoint differential operators. The formulation can be posed as follows:

$$\underset{\boldsymbol{c} \in \mathbb{R}^{ndofs}}{\text{minimize}} \quad \frac{1}{2}\langle \boldsymbol{c}; \boldsymbol{K}\boldsymbol{c} \rangle - \langle \boldsymbol{c}; \boldsymbol{f} \rangle, \quad \text{and} \tag{4.4.6a}$$

$$\text{subject to} \quad c_{\min}\mathbf{1} \preceq \boldsymbol{c} \preceq c_{\max}\mathbf{1}. \tag{4.4.6b}$$

In addition, if $\boldsymbol{K}$ is positive-definite the objective function becomes convex. The resulting optimization problem then belongs to the special case of convex quadratic programming for which sophisticated solvers exist.

Remark 4.4.1. *It should be mentioned that a quick fix to eliminate negative violations is through the so-called clipping procedure. However, this procedure is rather* ad hoc *and, more importantly, it is not variationally consistent. On the other hand, the proposed VI-based computational framework not only ensures non-negative solutions but also has a firm variational basis. We will also illustrate that the solutions under the proposed framework need not necessarily match the solution under the clipping procedure.*

4.4.1 Theoretical results in the discrete setting

In this chapter, we are interested in problems with two different cases of bound constraints. In the first case, we have both lower and upper bounds. In the second case, we have only the lower bound. The lower bound typically comes from the non-negative constraint, and the upper bound comes from maximum principles. We now discuss existence results for finite-dimensional VIs under the mentioned two cases of bound constraints.

We begin by noting that the feasible set $\mathcal{K}$ will be convex and closed for both the sets of bound constraints. In the first case, the feasible set will also be bounded, which makes the feasible set to be compact (which, in the context of Euclidean spaces, is equivalent to closed and bounded). We therefore deal with both the cases separately.

Theorem 4.4.2. *(Existence based on compactness of $\mathcal{K}$) If $\mathcal{K}$ is compact and convex, then a solution exists to the finite-dimensional VI* (4.4.5)*.*

Proof. A proof can be constructed using the Brouwer's fixed point theorem and can be found in [50]. □

Theorem 4.4.3. *(Existence based on positive-definiteness of* $\mathrm{sym}[\boldsymbol{K}]$*) If the symmetric part of the coefficient matrix $\boldsymbol{K}$ (i.e.,* $\mathrm{sym}[\boldsymbol{K}]$*) is positive-definite, a solution to the finite-dimensional VI* (4.4.5) *exists. (Note that the feasible set $\mathcal{K}$ need not be compact.)*

Proof. Let

$$\boldsymbol{g}(\boldsymbol{c}) := \boldsymbol{K}\boldsymbol{c} - \boldsymbol{f}. \tag{4.4.7}$$

The VI then becomes

$$\langle \boldsymbol{g}(\boldsymbol{c}); \widetilde{\boldsymbol{c}} - \boldsymbol{c} \rangle \geq 0 \quad \forall \widetilde{\boldsymbol{c}} \in \mathcal{K}. \tag{4.4.8}$$

Clearly, the function $\boldsymbol{g}(\boldsymbol{c})$ is continuous. Moreover, the function $\boldsymbol{g}(\boldsymbol{c})$ satisfies the following coercive condition

$$\frac{\langle \boldsymbol{g}(\boldsymbol{c}) - \boldsymbol{g}(\widetilde{\boldsymbol{c}}); \boldsymbol{c} - \widetilde{\boldsymbol{c}} \rangle}{\|\boldsymbol{c} - \widetilde{\boldsymbol{c}}\|} \to \infty, \tag{4.4.9}$$

as $\|\boldsymbol{c}\| \to \infty$. To wit, since $\mathrm{sym}[\boldsymbol{K}]$ is positive-definite and symmetric, the minimum eigenvalue λ_{min} is real and positive. One can then write

$$\lambda_{\mathrm{min}} \|\boldsymbol{c} - \widetilde{\boldsymbol{c}}\|^2 \leq (\boldsymbol{c} - \widetilde{\boldsymbol{c}}) \cdot \mathrm{sym}[\boldsymbol{K}]\,(\boldsymbol{c} - \widetilde{\boldsymbol{c}}) = (\boldsymbol{c} - \widetilde{\boldsymbol{c}}) \cdot \boldsymbol{K}\,(\boldsymbol{c} - \widetilde{\boldsymbol{c}}) = \langle \boldsymbol{g}(\boldsymbol{c}) - \boldsymbol{g}(\widetilde{\boldsymbol{c}}); \boldsymbol{c} - \widetilde{\boldsymbol{c}} \rangle. \tag{4.4.10}$$

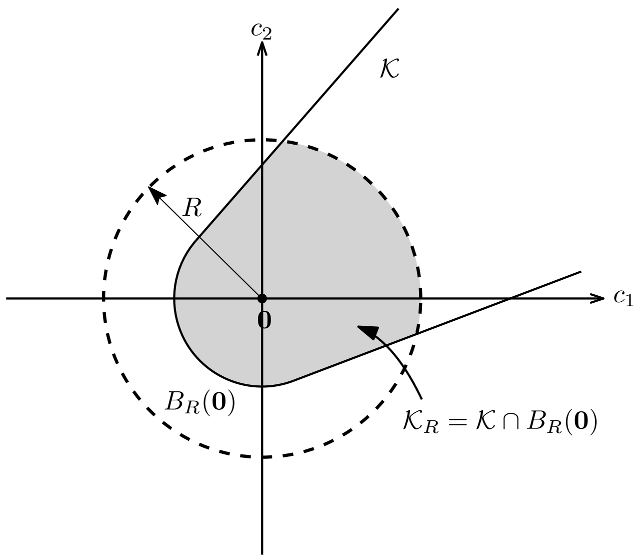


Figure 4.3: A pictorial description of $B_R(\mathbf{0})$ and $\mathcal{K}_R$. These sets are used in Theorem 4.4.4.

That is,

$$\lambda_{\min}\|\boldsymbol{c}-\widetilde{\boldsymbol{c}}\| \le \frac{\langle \boldsymbol{g}(\boldsymbol{c})-\boldsymbol{g}(\widetilde{\boldsymbol{c}});\boldsymbol{c}-\widetilde{\boldsymbol{c}}\rangle}{\|\boldsymbol{c}-\widetilde{\boldsymbol{c}}\|}. \tag{4.4.11}$$

We thus have shown that the function $\boldsymbol{g}(\boldsymbol{c})$ is continuous and coercive. Under such conditions, a solution exists to the VI (4.4.8) (e.g., see [128, 129]). □

We next present another existence theorem which is particularly useful when the feasible set is unbounded (for example, when we have only one of the bounds – either lower or upper bounds). Let $B_R(\mathbf{0})$ is a hypersphere of radius R centered at $\mathbf{0}$, and let $\mathcal{K}_R = \mathcal{K} \cap B_R(\mathbf{0})$ (see Figure 4.3). Clearly, $\mathcal{K}_R$ is bounded.

Theorem 4.4.4. *(Existence based on $\mathcal{K}_R$) A solution exists to the VI* (4.4.5) *on $\mathcal{K}$ (which need not be bounded) if and only if there exists $R > 0$ and a solution $\boldsymbol{c}^* \in \mathcal{K}_R$ that satisfies the following VI:*

$$\langle \boldsymbol{K}\boldsymbol{c}^*;\widetilde{\boldsymbol{c}}-\boldsymbol{c}^*\rangle \ge \langle f;\widetilde{\boldsymbol{c}}-\boldsymbol{c}^*\rangle \quad \forall \widetilde{\boldsymbol{c}} \in \mathcal{K}_R, \tag{4.4.12}$$

which is defined on a bounded set.

Proof. See [128]. $\square$

Theorem 4.4.5. *(Uniqueness) If the symmetric part of the coefficient matrix* $\boldsymbol{K}$ *is positive-definite, then the finite-dimensional VI* (4.4.5) *has a unique solution if it exists.*

Proof. On the contrary, assume that $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ are two different solutions of the VI (4.4.5). This implies that

$$\langle \boldsymbol{K}\boldsymbol{c}_1; \boldsymbol{v} - \boldsymbol{c}_1 \rangle \geq \langle \boldsymbol{f}; \boldsymbol{v} - \boldsymbol{c}_1 \rangle \quad \forall \boldsymbol{v} \in \mathcal{K} \quad \text{and} \tag{4.4.13}$$

$$\langle \boldsymbol{K}\boldsymbol{c}_2; \boldsymbol{v} - \boldsymbol{c}_2 \rangle \geq \langle \boldsymbol{f}; \boldsymbol{v} - \boldsymbol{c}_2 \rangle \quad \forall \boldsymbol{v} \in \mathcal{K}. \tag{4.4.14}$$

Since $\boldsymbol{c}_1, \boldsymbol{c}_2 \in \mathcal{K}$, choose $\boldsymbol{v} = \boldsymbol{c}_2$ in (4.4.13) and $\boldsymbol{v} = \boldsymbol{c}_1$ in (4.4.14). This results in

$$\langle \boldsymbol{K}\boldsymbol{c}_1; \boldsymbol{c}_2 - \boldsymbol{c}_1 \rangle \geq \langle \boldsymbol{f}; \boldsymbol{c}_2 - \boldsymbol{c}_1 \rangle \quad \text{and} \tag{4.4.15}$$

$$\langle \boldsymbol{K}\boldsymbol{c}_2; \boldsymbol{c}_1 - \boldsymbol{c}_2 \rangle \geq \langle \boldsymbol{f}; \boldsymbol{c}_1 - \boldsymbol{c}_2 \rangle. \tag{4.4.16}$$

Summing the above two inequalities and invoking the linearity in the second slot, we obtain

$$\langle \boldsymbol{K}(\boldsymbol{c}_1 - \boldsymbol{c}_2); \boldsymbol{c}_1 - \boldsymbol{c}_2 \rangle \leq \boldsymbol{0}, \tag{4.4.17}$$

which further implies that

$$\langle \mathrm{sym}[\boldsymbol{K}](\boldsymbol{c}_1 - \boldsymbol{c}_2); \boldsymbol{c}_1 - \boldsymbol{c}_2 \rangle \leq \boldsymbol{0}. \tag{4.4.18}$$

On the other hand, the positive-definiteness of $\mathrm{sym}[\boldsymbol{K}]$ and our assumption $\boldsymbol{c}_1 - \boldsymbol{c}_2 \neq \boldsymbol{0}$

imply that

$$\langle \mathrm{sym}[\boldsymbol{K}](\boldsymbol{c}_1 - \boldsymbol{c}_2); \boldsymbol{c}_1 - \boldsymbol{c}_2 \rangle > \boldsymbol{0}, \tag{4.4.19}$$

which contradicts the inequality given by equation (4.4.18). Hence, $\boldsymbol{c}_1 = \boldsymbol{c}_2$. □

Using the aforementioned general existence and uniqueness results for VIs, we now establish the existence and uniqueness of solutions under the proposed framework in the discrete setting.

Theorem 4.4.6. *(Well-posedness of the proposed framework) Unique solutions exist for the VIs from the GAL, SUPG and DG WFs under lower bounds (which arise from the non-negative constraint) and under both lower and upper bounds (which arise from maximum principles).*

Proof. First, it should be noted that the symmetric part of the coefficient matrices under the GAL and DG formulations are positive-definite. On the other hand, the stabilization term under the SUPG formulation does not guarantee that the symmetric part of the coefficient matrix to be positive-definite. It should also be noted that the stabilization term in equation (4.3.10) is $\mathcal{O}(h)$, where h is the characteristic mesh size. This means that there exists a critical mesh size, h_{crit}, such that if $h < h_{\mathrm{crit}}$ then the contribution from the residual terms to the coefficient matrix will be small, and the resulting symmetric part of the coefficient matrix will be positive-definite.

(*Existence.*) If both the lower and upper bounds are present, the feasible region will be compact. For this case of bound constraints, Theorem 4.4.2 establishes the existence of solutions for the VIs arising from all the three WFs (i.e., GAL, SUPG and DG). If only the lower bounds are present, Theorem 4.4.3 will ensure the existence of solutions for VIs arising from the GAL and DG formulations, and Theorem 4.4.4 will ensure the existence of solutions for the VIs arising from the SUPG formulation on a general mesh. Of course, if the mesh is adequately refined (i.e., $h < h_{\mathrm{crit}}$) then

Theorem 4.4.3 can also ensure the existence of a solution for the VIs arising under the SUPG formulation.

(*Uniqueness.*) Theorem 4.4.5 provides the uniqueness of solution for the VIs arising from the GAL and DG formulations. As discussed above, upon an adequate mesh refinement, sym$[\boldsymbol{K}]$ will be positive-definite under the SUPG formulation. On those meshes, Theorem 4.4.5 will provide the uniqueness of solutions for the VIs arising from the SUPG formulation. $\square$

4.4.2 Computer implementation details

In this chapter, the proposed QP and VI-based formulations for the GAL, SUPG, and DG formulations are implemented through the Firedrake project (see Appendix A for further details), but one can employ any other finite element library. The primary advantage of the Firedrake project is that it provides easy access to parallel solvers, specifically the PETSc and TAO libraries [12, 13, 123] which are built on top of Message Passing Interface (MPI) libraries. Appropriate iterative solvers and preconditioners are needed for large-scale problems, and the PETSc library provides the necessary data structures. The Conjugate Gradient (CG) method is used for symmetric problems like the diffusion equation whereas the Generalized Minimal Residual (GMRES) method is used for the non-symmetric advection-diffusion equation. *HYPRE*'s algebraic multi-grid package [93] is used as the preconditioner.

Solvers

The main ingredient of the proposed computational framework is to solve finite-dimensional VIs. There are several solvers available for solving these type of inequalities in a large-scale parallel environment. However, the performance of these solvers is problem-specific. It is, therefore, necessary to identify the best performing VI solver for our case, i.e., a solver enforcing maximum principles and the non-negative constraint. To this end, we consider the following VI and QP solvers available through the PETSc and TAO libraries:

1. Semi-smooth (VI - SS): TAO's implementation of the semi-smooth algorithm [109, 124] reformulates the MCP as a non-smooth system of equations using the Fischer-Burmeister function [54]. This function, $\phi : \mathbb{R}^2 \to \mathbb{R}$, is defined as

$$\phi(a, b) := \sqrt{a^2 + b^2} - a - b \quad \text{and} \tag{4.4.20a}$$

$$\phi(a, b) = 0 \quad \Leftrightarrow \quad a \geq 0,\ b \geq 0,\ ab = 0. \tag{4.4.20b}$$

   The reformulation of the MCP is handled component-wise, and the system of equations $\Phi(\boldsymbol{c}) = 0$ where $\Phi : \mathbb{R}^{ndofs} \to \mathbb{R}^{ndofs}$ is expressed as

$$\Phi_i(\boldsymbol{c}) := \begin{cases} \phi\left(c_i - c_{\min},\ h_i(\boldsymbol{c})\right) & \text{if } -\infty < c_{\min} < c_{\max} = \infty, \\ \phi\left(c_{\max} - c_i,\ -h_i(\boldsymbol{c})\right) & \text{if } -\infty = c_{\min} < c_{\max} < \infty, \\ \phi\left(c_i - c_{\min},\ \phi\left(c_{\max} - c_i,\ -h_i(\boldsymbol{c})\right)\right) & \text{if } -\infty < c_{\min} < c_{\max} < \infty, \\ -h_i(\boldsymbol{c}) & \text{if } -\infty = c_{\min} < c_{\max} = \infty, \\ c_{\min} - c_i & \text{if } -\infty < c_{\min} = c_{\max} < \infty. \end{cases} \tag{4.4.21}$$

   It should be noted that $\Phi(\boldsymbol{c})$ is not differentiable everywhere but it still satisfies a semi-smoothness property [117, 138, 139]. The above system of equations is used to compute a descent direction, and the solver finishes as soon as the natural merit function $\Psi(\boldsymbol{c}) := \frac{1}{2}\|\Phi(\boldsymbol{c})\|_2^2$ meets some level of tolerance. We also employ TAO's feasible line-search algorithm which ensures that the solution is within the bounds by using a projected Armijo line search [8].

2. Reduced-space active-set (VI - RS): The reduced-space active-set method selects an active-set and solves a reduced linear system of equations to calculate a

direction of the gradient descent. The active and inactive sets are, respectively, defined as:

$$\mathcal{A}(\boldsymbol{c}) := \{i \in \{1, ..., ndofs\} \;\Big|\; c_i = 0 \text{ and } h_i(\boldsymbol{c}) > 0\} \quad \text{and} \tag{4.4.22a}$$

$$\mathcal{I}(\boldsymbol{c}) := \{i \in \{1, ..., ndofs\} \;\Big|\; c_i > 0 \text{ or } h_i(\boldsymbol{c}) \leq 0\}. \tag{4.4.22b}$$

The active set $\mathcal{A}(\boldsymbol{c})$ represents regions where the lower bound is active thus the function value can be ignored, and the inactive set $\mathcal{I}(\boldsymbol{c})$ contains everything else. The descent direction of the active set is set to zero whereas the descent direction of the inactive set is approximated, and the solution is updated using a projected line search. Unlike the standard clipping procedure, this method is still variationally consistent because only the gradient descent is clipped and not the actual solution itself. As far as we know, there is little documentation on the theoretical and mathematical convergence properties for this particular algorithm, but the computational results from [15] demonstrate that this solver is robust and can handle a wide range of applications. For further implementation details of these two VI solvers, we defer all interested readers to [15, 123] and the references within.

3. Trust region Newton (QP - TRON): Unlike the previous two solvers, the trust region Newton method [102] is an active-set solver designed for large-scale minimization problems. It uses the gradient projection to generate a Cauchy step and the preconditioned CG with an incomplete Cholesky factorization to generate a descent direction. Each iteration of the TRON algorithm solves a reduced linear system containing variables that lie between the lower and upper bounds. The algorithm then applies a trust region to the conjugate gradients to ensure convergence. The algorithmic scalability and hardware performance of this solver has been thoroughly documented in [30], so the computational results

arising from QP - TRON serves primarily as a benchmark for comparison with the VI solvers.

We acknowledge that there may be several other QP and VI solvers which are not covered in this chapter. Nonetheless, the computational framework that we propose is algorithm-independent and platform-agnostic, so one is free to either employ different solvers or modify the above implementations of the QP and VI algorithms to cater to specific needs and applications.

An outline of the algorithm

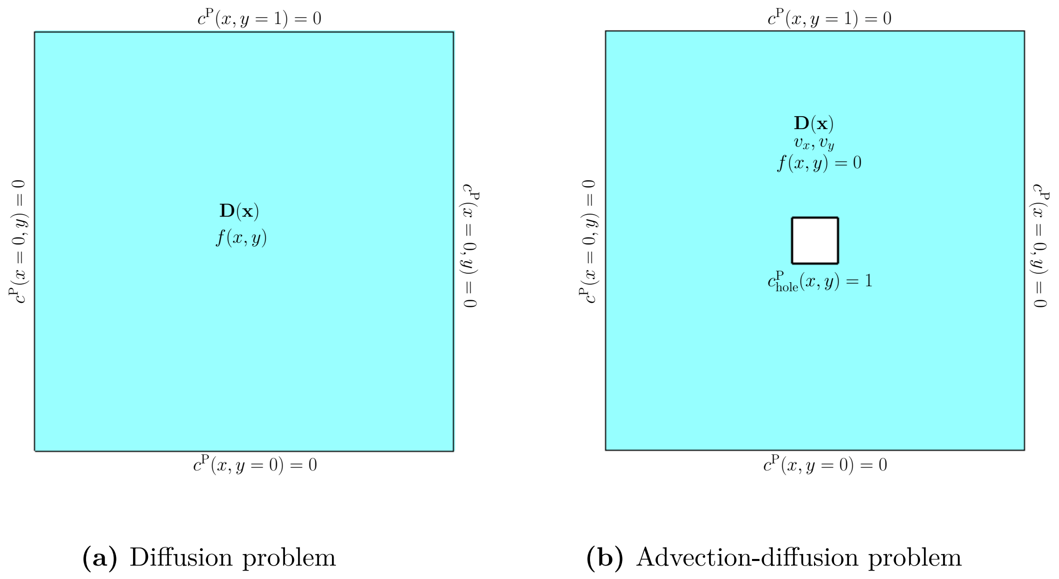
The performance of non-linear and optimization-based solvers depends on accurate initial guesses. To this end, we propose the following steps for the overall implementation of the proposed computational framework:

1. Assemble $\boldsymbol{K}$ and $\boldsymbol{f}$
2. Solve for $\boldsymbol{c}_0$.
3. Clip $\boldsymbol{c}_0$ and obtain $\boldsymbol{c}_{\mathrm{CLIP}}$. Formally

$$\boldsymbol{c}_{\mathrm{CLIP}} = \arg\min \frac{1}{2}\|\boldsymbol{c} - \boldsymbol{c}_0\|^2 \quad \text{subject to: } c_{\min}\mathbf{1} \preceq \boldsymbol{c} \preceq c_{\max}\mathbf{1}. \tag{4.4.23}$$

4. Solve the bounded constraint problem under the QP or VI framework with $\boldsymbol{c}_{\mathrm{CLIP}}$ as the initial guess.

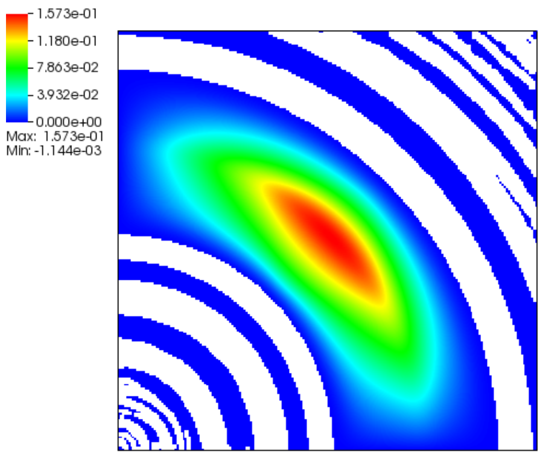
It should be emphasized that one need not solve equation (4.4.23) to implement the clipping procedure. Instead, one trims nodal values to meet the desired bounds. Since the governing equations are linear, $\boldsymbol{K}$ and $\boldsymbol{f}$ only need to be assembled once and are reused for the various QP and VI evaluation routines for Step 4. Python implementations of the VI - SS, VI - RS, and QP - TRON solvers leveraging petsc4py [42] capabilities are shown in Listings A.5, A.6, and A.7 respectively of Appendix A. For



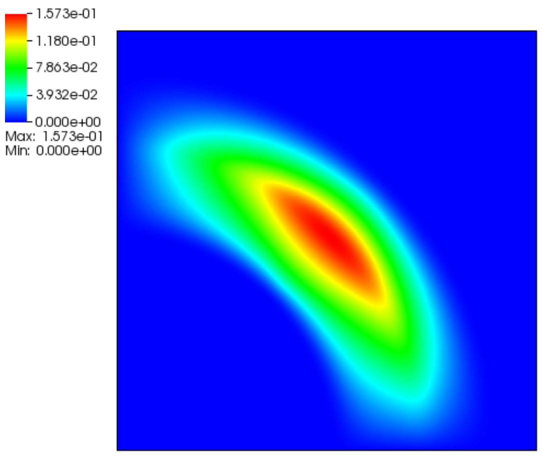
$c^{\mathrm{P}}(x, y=1)=0$
$c^{\mathrm{P}}(x=0, y)=0$
$\mathbf{D}(\mathbf{x})$
$f(x, y)$
$c^{\mathrm{P}}(x=0, y)=0$
$c^{\mathrm{P}}(x, y=0)=0$
(a) Diffusion problem
$c^{\mathrm{P}}(x, y=1)=0$
$c^{\mathrm{P}}(x=0, y)=0$
$\mathbf{D}(\mathbf{x})$
v_x, v_y
$f(x, y)=0$
$c^{\mathrm{P}}_{\mathrm{hole}}(x, y)=1$
$c^{\mathrm{P}}(x=0, y)=0$
$c^{\mathrm{P}}(x, y=0)=0$
(b) Advection-diffusion problem

Figure 4.4: 2D benchmarks: Pictorial description of the boundary value problems. Left figure contains 40,000 structured quadrilateral elements (40,401 nodes). Right figure contains 96,430 unstructured triangle elements (48,663 nodes).

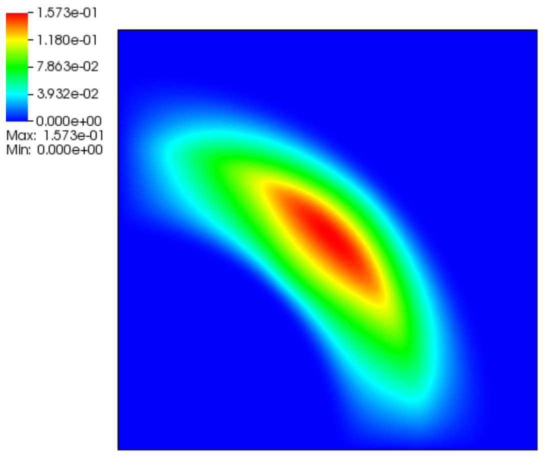
the steady-state 3D benchmarks in the next section, the KSP relative tolerance is set to 10^{-7} for the solver in Step 2, whereas the KSP relative tolerance for approximating the gradient descents in Step 4 is set to 10^{-3}. It was shown in Chapter 3 that relaxing the tolerance requires more non-linear iterations but lessens the overall solve time. Relaxing the tolerance also lessens the arithmetic intensity where the performance is governed by the memory bandwidth thus making it less likely to achieve good speedup on a shared compute node. In other words, the parallel efficiency of the QP and VI solvers are likely to be worse than solving the WF with standard KSP convergence tolerances. The absolute convergence tolerances for the QP and VI solvers are set to 10^{-8} although it should be mentioned that the optimal values depends on the application at hand. All 3D simulations computations are conducted on Intel Xeon E5-2680v2 processors where each MPI process is restricted to a single core and mapped in a round-robin order.



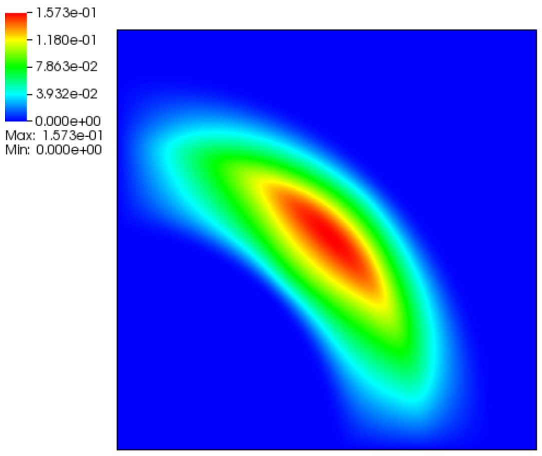
(a) $\boldsymbol{c}_{\mathrm{GAL}}$



(b) $\boldsymbol{c}_{\mathrm{TRON}}$



(c) $\boldsymbol{c}_{\mathrm{SS}}$



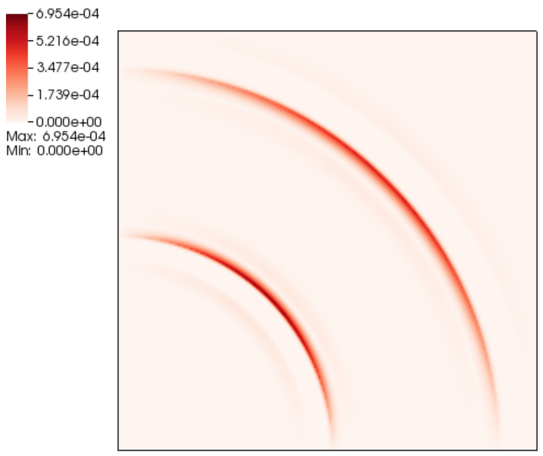
(d) $\boldsymbol{c}_{\mathrm{RS}}$

Figure 4.5: 2D diffusion: concentrations under the Galerkin ($\boldsymbol{c}_{\mathrm{GAL}}$), TRON ($\boldsymbol{c}_{\mathrm{TRON}}$), semi-smooth ($\boldsymbol{c}_{\mathrm{SS}}$), and reduced-space active-set ($\boldsymbol{c}_{\mathrm{RS}}$) methods where the white regions represent negative concentrations.

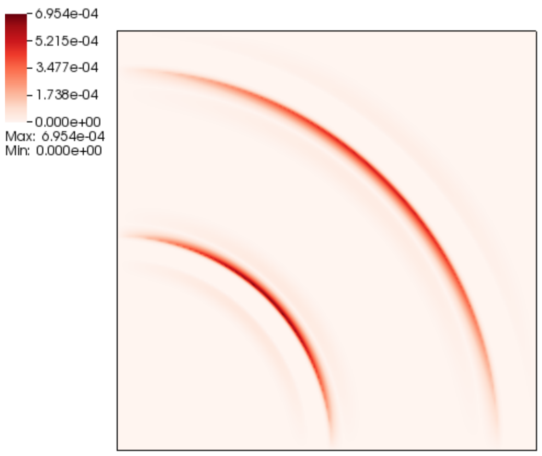
4.5 NUMERICAL RESULTS FOR STEADY-STATE RESPONSE

4.5.1 2D benchmarks

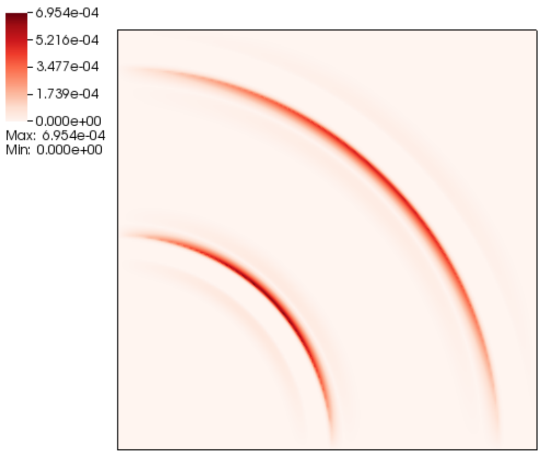
We now examine 2D problems in order to demonstrate the effectiveness of the proposed computational algorithms for ensuring discrete maximum principles and



(a) $\|\boldsymbol{c}_{\mathrm{CLIP}} - \boldsymbol{c}_{\mathrm{TRON}}\|$



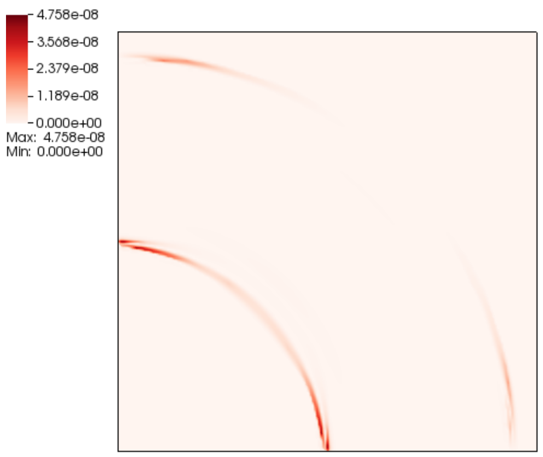
(b) $\|\boldsymbol{c}_{\mathrm{CLIP}} - \boldsymbol{c}_{\mathrm{SS}}\|$



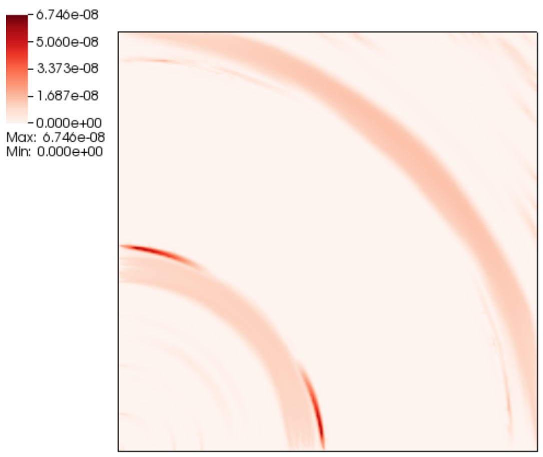
(c) $\|\boldsymbol{c}_{\mathrm{CLIP}} - \boldsymbol{c}_{\mathrm{RS}}\|$

Figure 4.6: 2D diffusion: absolute difference in concentrations between the clipped solution and the non-negative solution.

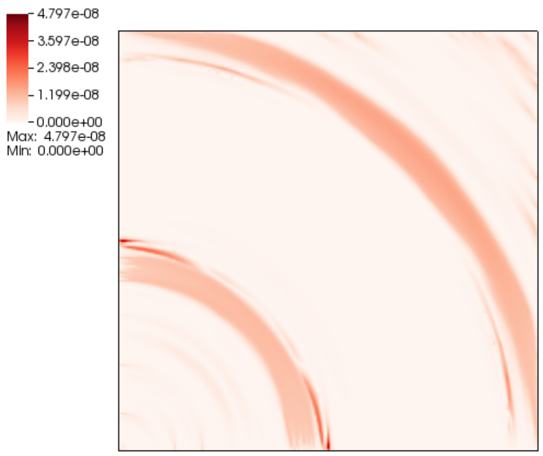
the non-negative constraint. Only the GAL and SUPG formulations are employed in this numerical study. First, let us consider the pure diffusion equation on a bi-unit square: $\Omega := (0, 1) \times (0, 1)$ as shown in Figure 4.4a. The following heterogeneous and



(a) $\|\boldsymbol{c}_{\text{TRON}} - \boldsymbol{c}_{\text{SS}}\|$



(b) $\|\boldsymbol{c}_{\text{TRON}} - \boldsymbol{c}_{\text{RS}}\|$



(c) $\|\boldsymbol{c}_{\text{SS}} - \boldsymbol{c}_{\text{RS}}\|$

Figure 4.7: 2D diffusion: absolute difference in concentrations between the various non-negative methodologies

anisotropic diffusivity tensor similar to the one considered in [135] is used:

$$\mathbf{D}(\mathbf{x}) = \begin{pmatrix} y^2 + \epsilon x^2 & -(1-\epsilon)xy \\ -(1-\epsilon)xy & x^2 + \epsilon y^2 \end{pmatrix}, \tag{4.5.1}$$

where $\epsilon = 10^{-4}$. The forcing function is defined as $f(x, y) = 1$ if $(x, y) \in \left[\frac{3}{8}, \frac{5}{8}\right] \times \left[\frac{3}{5}, \frac{5}{8}\right]$ and zero elsewhere. Homogeneous boundary conditions are applied on all four sides of the domain. Numerical solutions under the GAL, VI - SS, VI - RS, and QP - TRON methods with uniform quadrilateral elements of h-size $= 1/200$ are shown in Figure 4.5. All three non-negative solvers successfully eliminate negative concentrations, and the absolute difference plots in Figure 4.6 show that their results are quite different than from the one arising from the standard clipping procedure. Moreover, the absolute differences between the various QP and VI solvers, as seen from Figure 4.7, are extremely small and suggest that the QP and VI solvers have similar numerical accuracy.

Next we consider the advection-diffusion problem under the SUPG formulation where only VI - SS and VI - RS methods are applicable for enforcing the maximum principle and the non-negative constraint. Consider a bi-unit square: $\Omega := (0, 1) \times (0, 1)$ with a square hole of dimension $\left[\frac{4}{9}, \frac{5}{9}\right] \times \left[\frac{4}{9}, \frac{5}{9}\right]$ as shown in Figure 4.4b. The mesh is discretized into 96,430 unstructured triangular elements and 48,663 vertices. Homogeneous boundary conditions are applied on the outside boundary, and a Dirichlet boundary value $c^{\mathrm{p}}(\mathbf{x}) = 1$ is applied on the interior boundary Γ^{hole}. The velocity vector field $\mathbf{v}(\mathbf{x})$ is characterized by the following

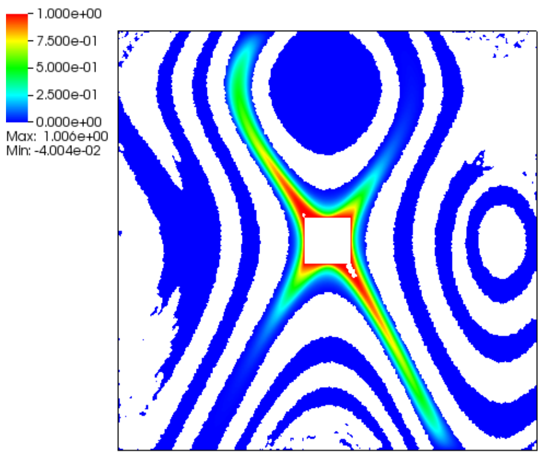
$$v_x = \cos(2\pi y^2) \quad \text{and} \tag{4.5.2a}$$

$$v_y = \sin(2\pi x) + \cos(2\pi x^2), \tag{4.5.2b}$$

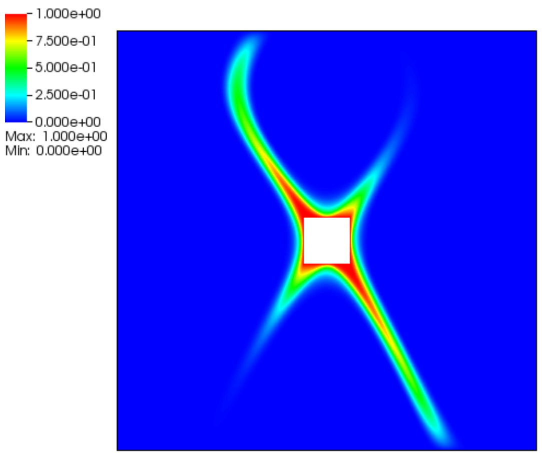
and the diffusivity tensor $\mathbf{D}(\mathbf{x})$ for this problem is the dispersion tensor

$$\mathbf{D}(\mathbf{x}) = (\alpha_T \|\mathbf{v}\| + D_M)\,\mathbf{I} + (\alpha_L - \alpha_T)\frac{\mathbf{v} \otimes \mathbf{v}}{\|\mathbf{v}\|}, \tag{4.5.3}$$

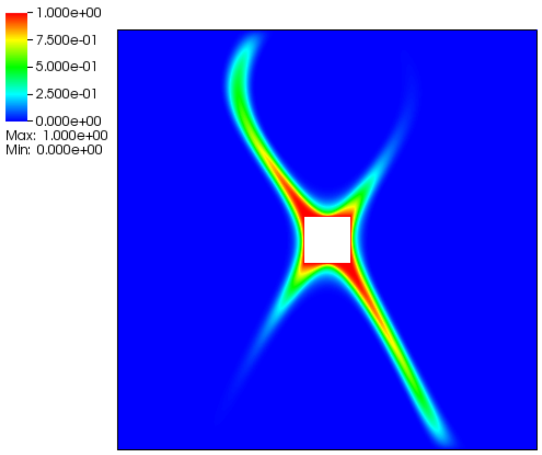
where $\alpha_L = 10^{-1}$, $\alpha_T = 10^{-5}$, and $D_M = 10^{-9}$ denote the longitudinal dispersivity, transverse dispersivity and molecular diffusivity, respectively. Figure 4.8 depicts the



(a) $\boldsymbol{c}_{\text{SUPG}}$



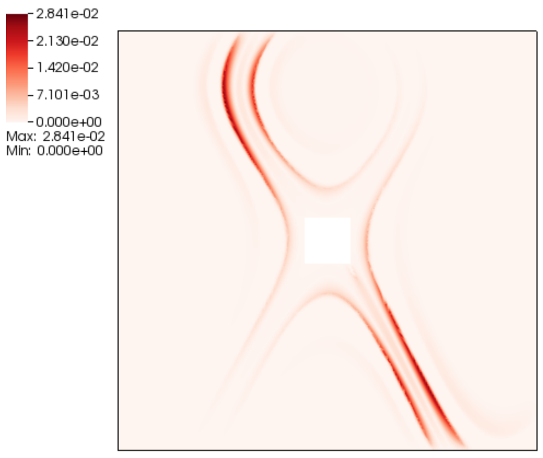
(b) $\boldsymbol{c}_{\text{SS}}$



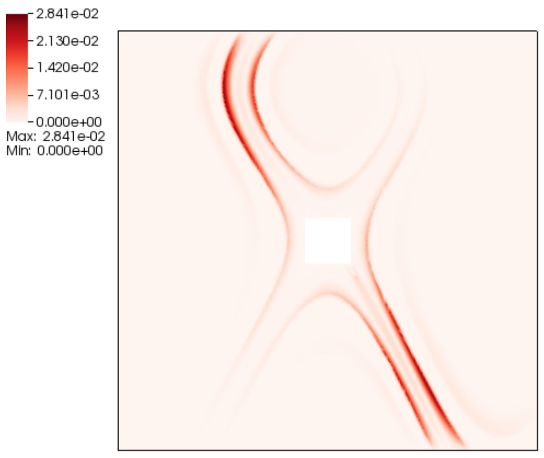
(c) $\boldsymbol{c}_{\text{RS}}$

Figure 4.8: 2D advection-diffusion: concentrations under the SUPG ($\boldsymbol{c}_{\text{SUPG}}$), semi-smooth ($\boldsymbol{c}_{\text{SS}}$), and reduced-space active-set ($\boldsymbol{c}_{\text{RS}}$) methods where the white regions represent negative concentrations.

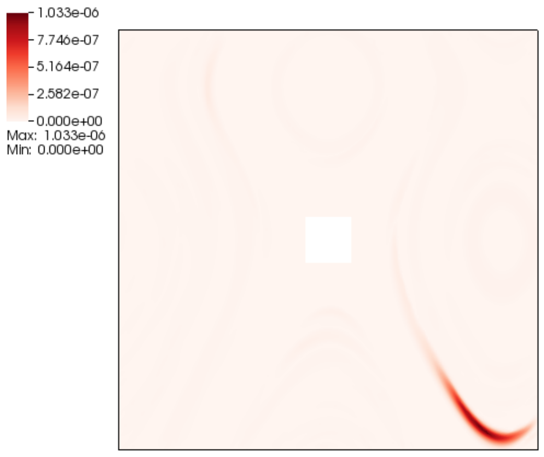
numerical solutions under the SUPG, VI - SS, and VI - RS formulations. We see that the SUPG formulation results in negative concentrations as well as concentrations greater than the maximum prescribed boundary condition whereas the two VI solvers successfully correct these concentrations. *The absolute difference plots, as seen in Figure 4.9, indicate that the VI are also similar to one another and differ from the*



(a) $\|\boldsymbol{c}_{\mathrm{CLIP}} - \boldsymbol{c}_{\mathrm{SS}}\|$



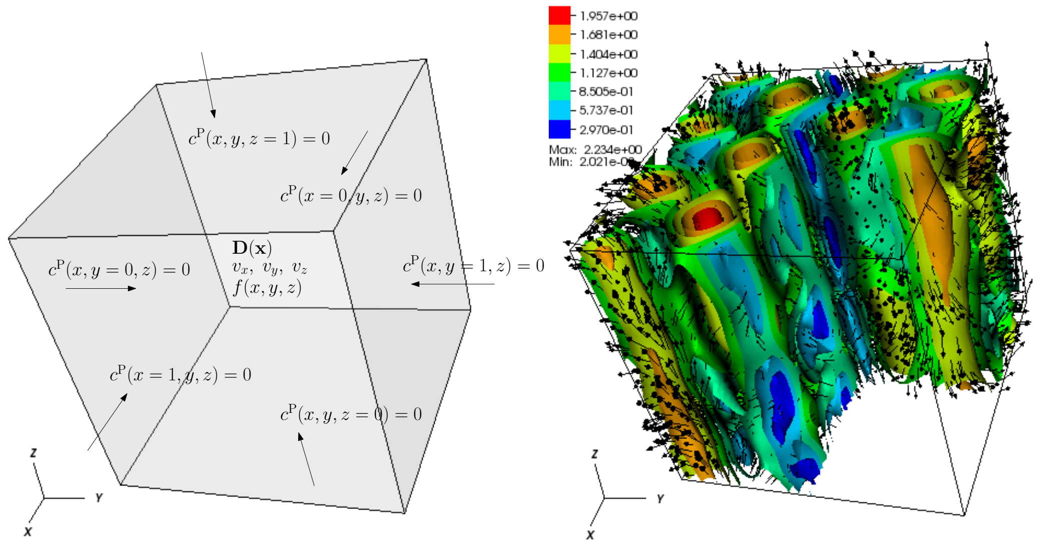
(b) $\|\boldsymbol{c}_{\mathrm{CLIP}} - \boldsymbol{c}_{\mathrm{RS}}\|$



(c) $\|\boldsymbol{c}_{\mathrm{SS}} - \boldsymbol{c}_{\mathrm{RS}}\|$

Figure 4.9: 2D advection-diffusion: absolute difference in concentrations between the clipped and non-negative solutions.

clipping procedure. These 2D benchmarks suggest that the QP and VI solvers are accurate alternatives to the clipping procedure for satisfying the discrete maximum principle and the non-negative constraint. Listings A.1 and A.2 contain the Firedrake project files for solving the GAL and SUPG formulations, respectively.



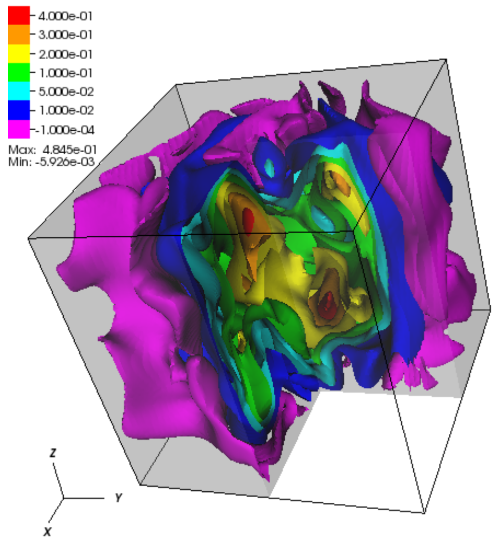
$c^{\mathrm{P}}(x, y, z = 1) = 0$
$c^{\mathrm{P}}(x = 0, y, z) = 0$
$\mathbf{D}(\mathbf{x})$
$v_x,\ v_y,\ v_z$
$f(x, y, z)$
$c^{\mathrm{P}}(x, y = 0, z) = 0$
$c^{\mathrm{P}}(x, y = 1, z) = 0$
$c^{\mathrm{P}}(x = 1, y, z) = 0$
$c^{\mathrm{P}}(x, y, z = 0) = 0$
Z
Y
X
1.957e+00
1.681e+00
1.404e+00
1.127e+00
8.505e-01
5.737e-01
2.970e-01
Max: 2.234e+00
Min: 2.021e-0

Figure 4.10: 3D benchmarks: Left figure contains a pictorial description of the boundary value problem. Right figure contains the corresponding velocity contour and vector field for the ABC flow.

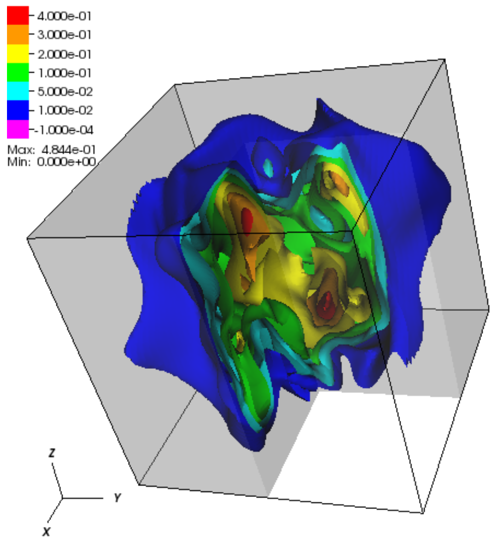
4.5.2 3D benchmark

We now consider a 3D problem designed to capture two particular aspects that may arise in large-scale applications: 1) chaotic advection, which is pervasive in many porous media applications [94], and 2) random point sources, which in subsurface remediation problems are the sites where potential contaminant leaks occur. Predictive modeling involving such important aspects require numerical methodologies that are not only accurate but also fast and scalable in a parallel environment. Herein, our goal is to study the computational performance of the various QP and VI solvers under the GAL, SUPG, and DG formulations.

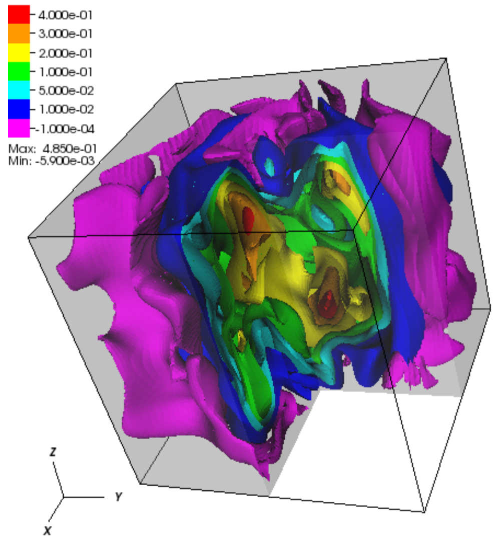
Consider a unit cube domain as shown in Figure 4.10 with chaotic advection



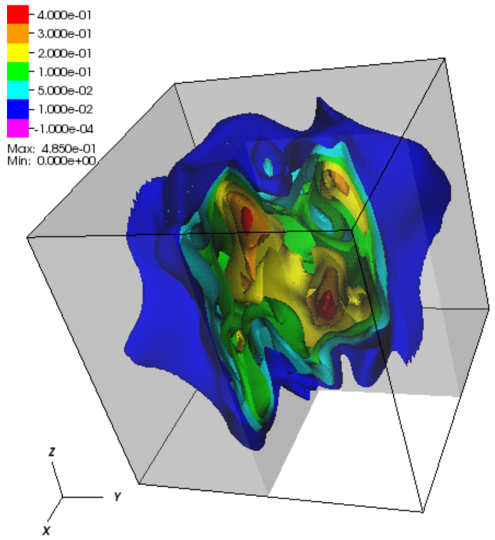
(a) GAL



(b) GAL with VI - SS



(c) DG



(d) DG with VI - SS

Figure 4.11: 3D diffusion: 3D contours of the concentrations for the GAL and DG formulations with and without and VI - SS for h-size $= 1/80$, where the purple contours represent regions with negative concentrations.

flow characterized by the Arnold-Beltrami-Childress (ABC) flow [166, 44]

$$v_x = 0.3\sin(2\pi z) + \cos(3\pi y), \tag{4.5.4a}$$

$$v_y = 0.65\sin(2\pi x) + 0.3\cos(5\pi z), \quad \text{and} \tag{4.5.4b}$$

$$v_z = \sin(4\pi y) + 0.65\cos(6\pi y). \tag{4.5.4c}$$

Table 4.1: 3D diffusion: minimum and maximum concentrations for various level of mesh refinement under the GAL formulation.

h-size	Min.	Max.	% degrees-of-freedom violated
1/10	-0.0224497	0.368322	280/1,331 $\rightarrow$ 21.0%
1/20	-0.0071611	0.339679	2,462/9,261 $\rightarrow$ 26.6%
1/30	-0.0083804	0.481598	8,449/29,791 $\rightarrow$ 28.4%
1/40	-0.0062918	0.378390	20,195/68,921 $\rightarrow$ 29.3%
1/50	-0.0067679	0.477119	39,500/132,651 $\rightarrow$ 29.8%
1/60	-0.0072030	0.518469	68,161/226,981 $\rightarrow$ 30.0%
1/70	-0.0066007	0.498127	109,554/357,911 $\rightarrow$ 30.6%
1/80	-0.0059264	0.484484	160,925/531,441 $\rightarrow$ 30.3%

Table 4.2: 3D diffusion: minimum and maximum concentrations for various level of mesh refinement under the DG formulation.

h-size	Min.	Max.	% degrees-of-freedom violated
1/10	-0.0226040	0.372831	3,704/8,000 $\rightarrow$ 46.3%
1/20	-0.0071913	0.341955	27,496/64,000 $\rightarrow$ 43.0%
1/30	-0.0082811	0.483264	91,176/216,000 $\rightarrow$ 42.2%
1/40	-0.0062341	0.379389	213,000/512,000 $\rightarrow$ 41.6%
1/50	-0.0067168	0.478146	410,976/1,000,000 $\rightarrow$ 41.1%
1/60	-0.0071682	0.519338	702,504/1,728,000 $\rightarrow$ 40.7%
1/70	-0.0065727	0.498775	1,114,856/2,744,000 $\rightarrow$ 40.6%
1/80	-0.0058998	0.485012	1,624,496/4,096,000 $\rightarrow$ 39.7%

For this problem, we shall also let $\mathbf{D}(\mathbf{x})$ denote the dispersion tensor as shown in equation (4.5.3) where $\alpha_L = 10^{-1}$, $\alpha_T = 10^{-5}$, and $D_M = 10^{-9}$. All six faces of the cube have homogeneous boundary conditions, and the following forcing function

Table 4.3: 3D diffusion: single core wall-clock time on an Intel Xeon E5-2680v2 server and number of solver iterations (KSP, VI, or QP) for various levels of mesh refinement under the GAL formulation.

h-size	GAL		VI - SS		VI - RS		QP - TRON	
	time (s)	iters	time (s)	iters	time (s)	iters	time (s)	iters
1/10	0.003	9	0.027	5	0.008	2	0.007	2
1/20	0.036	15	0.477	12	0.147	5	0.135	5
1/30	0.165	20	2.624	18	0.765	7	0.650	6
1/40	0.525	24	7.576	20	2.246	8	1.758	6
1/50	1.293	28	21.49	27	5.381	9	5.330	9
1/60	2.556	31	43.72	30	12.01	11	12.20	11
1/70	4.747	35	76.21	31	18.27	10	17.81	9
1/80	7.962	39	140.7	37	36.40	13	38.06	13

consisting of 8 randomly located point sources is used throughout the domain

$$f(x,y,z) = \begin{cases} 1 & \text{if } (x,y,z) \in [0.4,0.2,0.1] \times [0.5,0.3,0.2], \\ 1 & \text{if } (x,y,z) \in [0.8,0.4,0.2] \times [0.9,0.5,0.3], \\ 1 & \text{if } (x,y,z) \in [0.5,0.7,0.3] \times [0.6,0.8,0.4], \\ 1 & \text{if } (x,y,z) \in [0.3,0.5,0.2] \times [0.4,0.6,0.3], \\ 1 & \text{if } (x,y,z) \in [0.5,0.2,0.6] \times [0.6,0.3,0.7], \\ 1 & \text{if } (x,y,z) \in [0.6,0.5,0.7] \times [0.7,0.6,0.8], \\ 1 & \text{if } (x,y,z) \in [0.4,0.7,0.8] \times [0.5,0.8,0.9], \\ 1 & \text{if } (x,y,z) \in [0.1,0.4,0.7] \times [0.2,0.5,0.8], \quad \text{and} \\ 0 & \text{otherwise.} \end{cases} \tag{4.5.5}$$

To understand the parallel and algorithmic scalability of the QP and VI solvers, various levels of mesh refinement are considered, ranging from 1,331 to 1,030,301 degrees-of-freedom for the GAL/SUPG formulations and ranging from 8,000 to 4,096,000 degrees-of-freedom for the DG formulations. Up to 16 MPI processes are used to study the weak-scaling and strong-scaling potential of these solvers.

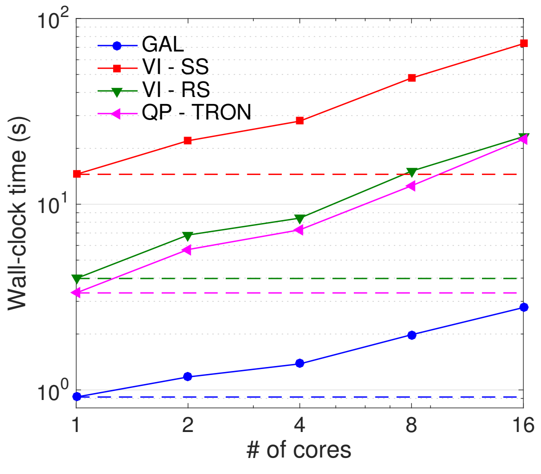
Figure 4.11 depicts the GAL and DG solutions for the diffusion equation with

Table 4.4: 3D diffusion: single core wall-clock time on an Intel Xeon E5-2680v2 server and number of solver iterations (KSP, VI, or QP) for various levels of mesh refinement under the DG formulation.

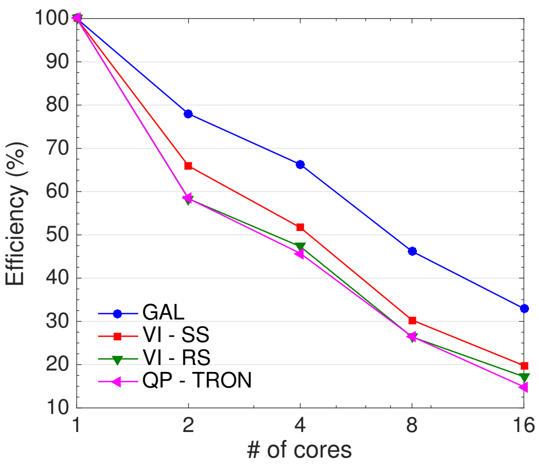
h-size	DG		VI - SS		VI - RS		QP - TRON	
	time (s)	iters	time (s)	iters	time (s)	iters	time (s)	iters
1/10	0.030	10	0.748	12	0.221	5	0.186	5
1/20	0.446	14	1.410	20	4.528	8	3.715	7
1/30	2.148	18	73.52	27	23.85	10	22.64	10
1/40	6.278	21	251.5	34	69.33	11	70.77	11
1/50	14.29	24	623.6	39	171.7	13	170.8	12
1/60	28.25	27	1290	45	360.1	15	388.6	15
1/70	51.95	31	2560	51	620.2	16	639.0	15
1/80	85.53	34	5049	54	1107	19	1291	17

and without VI - SS. It can be seen from the figures that negative concentrations are present regardless which finite element formulation is used. Tables 4.1 and 4.2 indicate that negative concentrations arise for the GAL and DG formulations, respectively, even as h-size is refined. It is interesting to note that the DG formulation not only has more degrees-of-freedom but has more regions with negative concentrations than its GAL counterpart. Using the initial guess solver from Step 2 of the proposed framework in 4.4.2 as a baseline for comparison, Tables 4.3 and 4.4 demonstrate how the wall-clock time and number of KSP/VI/QP solver iterations vary with h-refinement under a single MPI process. It should be noted that the timings for the QP and VI solvers consider both the assembly of the data structures as well as the actual solver. The heterogeneous nature of the problem causes the number of solvers iterations to increase with problem size, but the iteration counts begin to stabilize as the problem gets bigger. The VI - RS method outperforms VI - SS in both wall-clock time and VI iterations but has similar performance to QP - TRON.

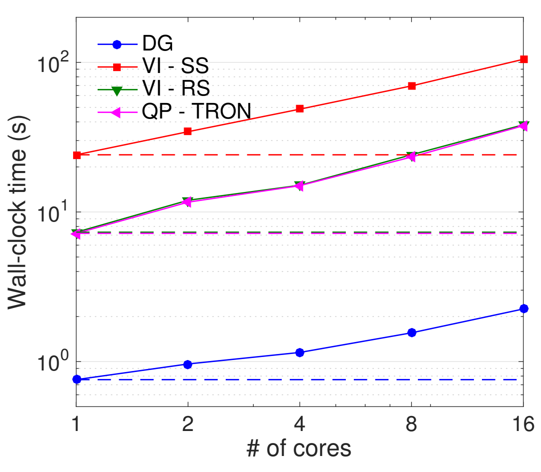
Next we perform weak-scaling studies to investigate how increasing both problem size and number of MPI processes affects the performance of the VI and QP solvers. Each MPI process will handle approximately 100k degrees-of-freedom so the h-sizes for the GAL case are 1/46, 1/58, 1/73, 1/92, and 1/116 for 1, 2, 4, 8, and 16 processes



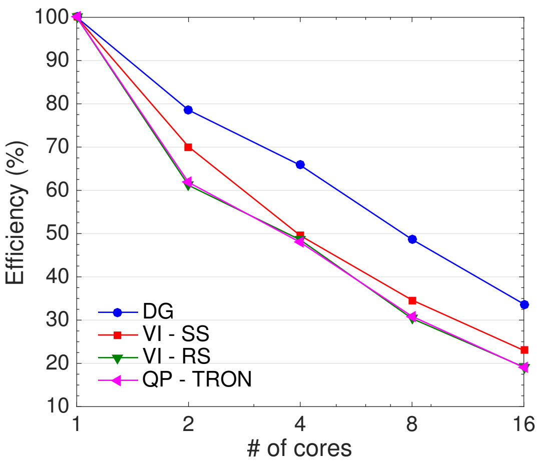
(a) GAL - solve time



(b) GAL - parallel efficiency



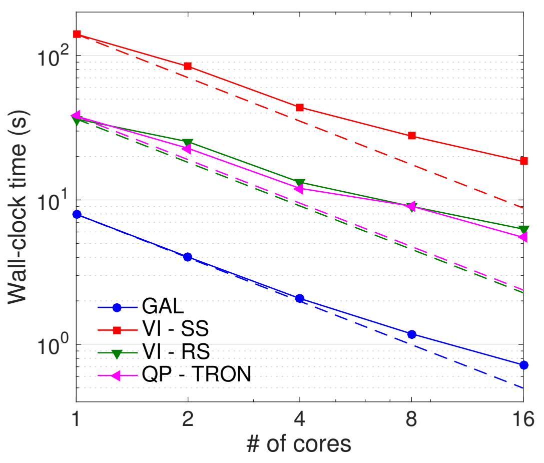
(c) DG - solve time



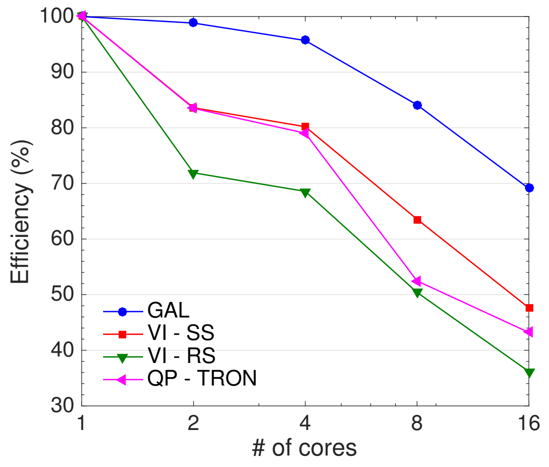
(d) DG - parallel efficiency

Figure 4.12: 3D diffusion: weak-scaling plots with approximately 100k degrees-of-freedom per core and the corresponding parallel efficiencies on a single Intel Xeon E5-2680v2 server.

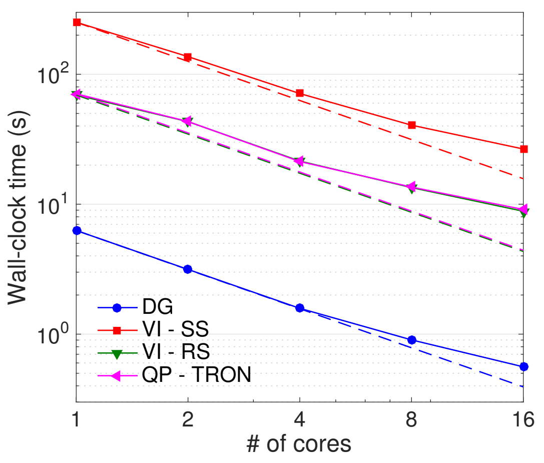
respectively whereas the h-sizes for the DG case are $1/23$, $1/29$, $1/37$, $1/46$, and $1/58$ for 1, 2, 4, 8, and 16 processes respectively. Figure 4.12 contains the scaling plots as well as the parallel efficiencies in the weak sense under the GAL and DG formulations. These plots suggest that the QP and VI methodologies are not linearly proportional to problem size (increasing solver iterations with h-size as seen from Tables 4.3 and 4.4). Furthermore, the lower KSP relative tolerance for the gradient descent computations make the solver more sensitive to the memory-bandwidth, meaning that performance



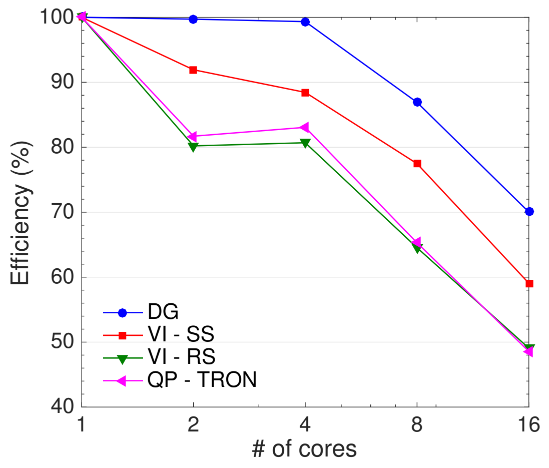
(a) GAL - solver time



(b) GAL - parallel efficiency



(c) DG - solver time

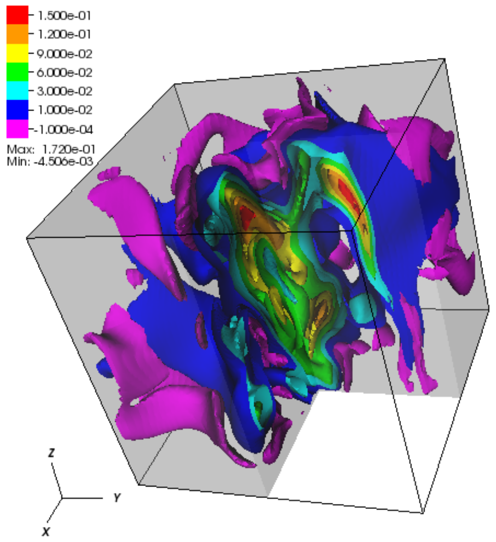


(d) DG - parallel efficiency

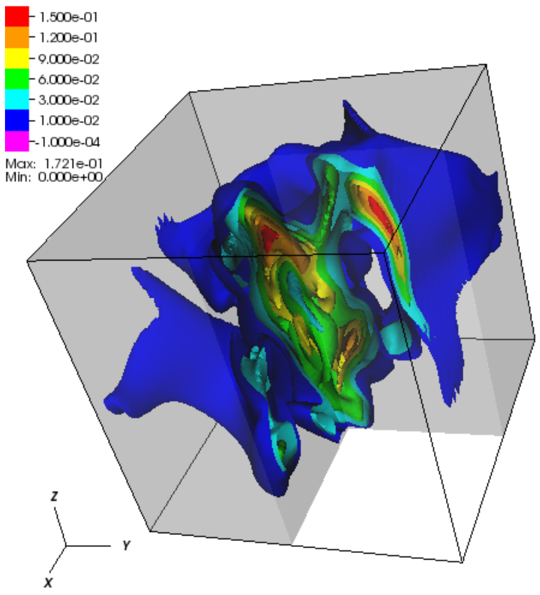
Figure 4.13: 3D diffusion: strong-scaling plots for approximately 500k degrees of freedom (h-size $= 1/80$ and $1/40$ for GAL and DG respectively) and the corresponding parallel efficiencies on a single Intel Xeon E5-2680v2 server.

can degrade as the compute nodes become populated with more MPI processes (see Sections 4 and 5 of [30] for a more thorough discussion).

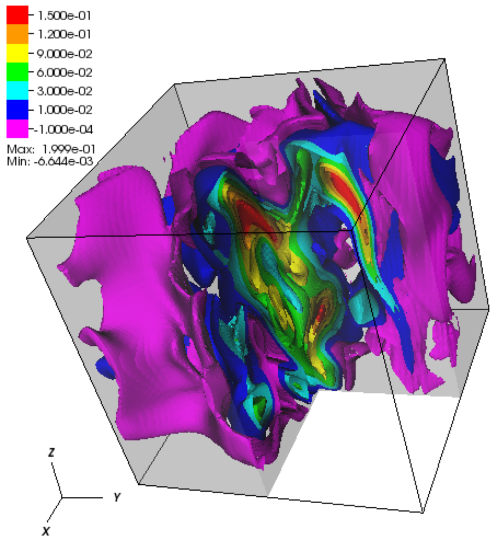
However, the weak-scaling plots alone make it difficult to distinguish whether parallel performance deteriorates due to communication overhead or suboptimal algorithmic convergence. To better understand why parallel performance degrades as the number of MPI processes increases, we conduct strong-scaling studies by setting the h-size to $1/80$ and $1/40$ for the GAL and DG formulations respectively (roughly 500k



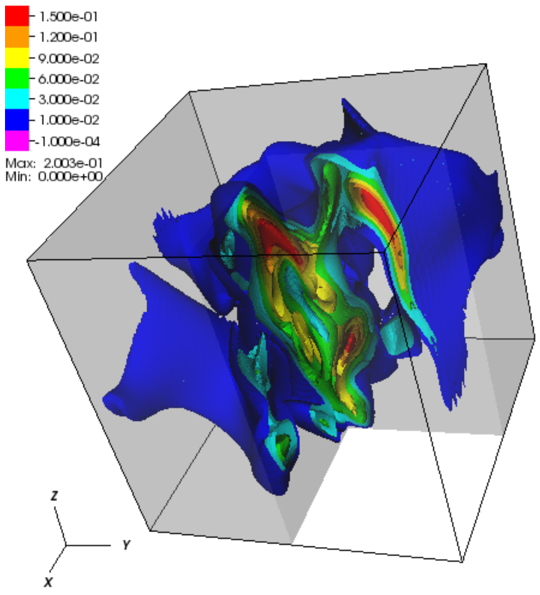
(a) SUPG



(b) SUPG with VI - SS



(c) DG



(d) DG with VI - SS

Figure 4.14: 3D advection-diffusion: 3D contours of the concentrations for the SUPG and DG formulations with and without and VI - SS for h-size $= 1/80$, where the purple contours represent regions with negative concentrations.

degrees-of-freedom) and study how increasing the number of MPI processes (hence communication overhead) affects the parallel performance. Figure 4.13 contains the strong-scaling plots, and we see that the QP and VI solvers still do not scale as well.

Table 4.5: 3D advection-diffusion: minimum and maximum concentrations for various level of mesh refinement under the SUPG formulation.

h-size	Min.	Max.	% degrees-of-freedom violated
1/10	-0.0135676	0.187489	212/1,331 → 15.9%
1/20	-0.0068733	0.180922	2,323/9,261 → 25.1%
1/30	-0.0091657	0.210942	7,964/29,791 → 26.7%
1/40	-0.0055686	0.171690	18,235/68,921 → 26.5%
1/50	-0.0064795	0.185440	35,221/132,651 → 26.6%
1/60	-0.0063168	0.189047	61,171/226,981 → 26.9%
1/70	-0.0053682	0.179675	99,668/357,911 → 27.8%
1/80	-0.0045065	0.172049	147,462/531,441 → 27.7%

Table 4.6: 3D advection-diffusion: minimum and maximum concentrations for various level of mesh refinement under the DG formulation.

h-size	Min.	Max.	% degrees-of-freedom violated
1/10	-0.0151514	0.259127	4,976/8,000 → 62.2%
1/20	-0.0162537	0.211295	37,464/64,000 → 58.5%
1/30	-0.0137824	0.237722	120,296/216,000 → 55.7%
1/40	-0.0067079	0.186956	276,832/512,000 → 54.1%
1/50	-0.0057574	0.203852	526,080/1,000,000 → 52.6%
1/60	-0.0065627	0.203093	891,768/1,728,000 → 51.6%
1/70	-0.0069389	0.193418	1,410,208/2,744,000 → 51.4%
1/80	-0.0066445	0.199912	2,069,752/4,096,000 → 50.5%

Regardless of the finite element formulation used, the QP and VI - RS methods have roughly the same strong-scaling performance whereas the VI - SS method has slightly better strong-scaling.

For the advection-diffusion equation, the same problem is considered but advection due to the ABC flow is now taken into account. A Firedrake project implementation of the DG formulation can be found in Listing A.3. Like the diffusion equation, the advection-diffusion equation also exhibits negative concentrations as seen from Figure 4.14. Table 4.5 depicts violations under the SUPG formulation to be no greater than 30% whereas the DG formulation exhibits huge violations as seen from Table 4.6. Moreover, the single MPI process metrics shown in Tables 4.7 and 4.8 clearly indicate that the advection-diffusion equations are generally more time consuming to solve than its diffusion counterpart. These metrics tell us that the VI

Table 4.7: 3D advection-diffusion: single core wall-clock time on an Intel Xeon E5-2680v2 server and number of linear (KSP) or nonlinear (VI) solve iterations for various levels of mesh refinement under the SUPG formulation.

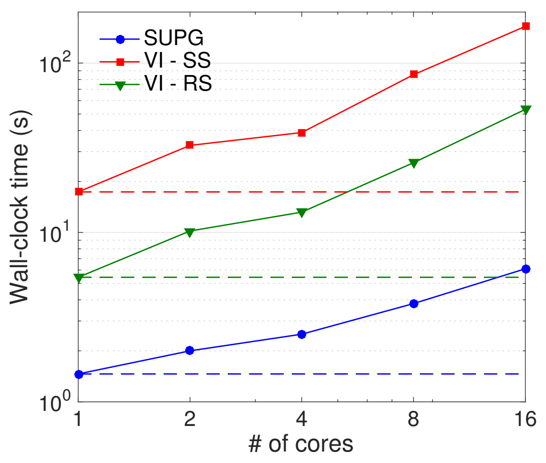
h-size	SUPG		VI - SS		VI - RS	
	time (s)	iters	time (s)	iters	time (s)	iters
1/10	0.003	9	0.028	5	0.009	2
1/20	0.045	16	0.504	11	0.146	4
1/30	0.221	22	2.525	14	0.710	5
1/40	0.768	29	10.21	20	2.592	6
1/50	2.019	35	27.27	23	8.842	10
1/60	4.530	43	65.61	30	19.20	10
1/70	8.178	47	117.3	28	34.54	11
1/80	14.70	55	229.4	34	66.07	12

Table 4.8: 3D advection-diffusion: single core wall-clock time on an Intel Xeon E5-2680v2 server time and number of linear (KSP) or nonlinear (VI) solve iterations for various levels of mesh refinement under the DG formulation.

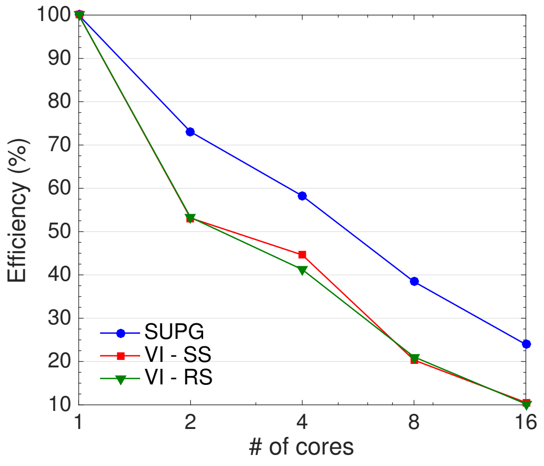
h-size	DG		VI - SS		VI - RS	
	time (s)	iters	time (s)	iters	time (s)	iters
1/10	0.031	10	0.807	13	0.202	5
1/20	0.459	14	14.46	21	3.890	8
1/30	2.352	19	73.84	27	19.56	10
1/40	6.819	22	251.3	35	56.94	11
1/50	15.49	25	629.1	40	137.1	13
1/60	30.65	28	1387	47	292.3	15
1/70	57.56	32	2267	51	570.0	18
1/80	97.15	36	7105	60	917.2	18

iterations also begin to stabilize as the problem size increases and that the VI - RS method is faster than the VI - SS method for the advection-diffusion equation.

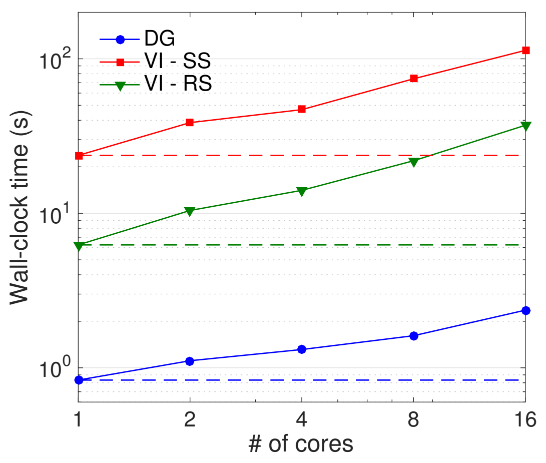
The weak-scaling plots, as seen from Figure 4.15, indicate that the VI solvers are identical in performance, but the strong-scaling plots from Figure 4.16 suggest that VI - SS has slightly better parallel speedup than VI - RS. These steady-state numerical experiments suggest that the VI - RS is the preferred methodology for solving large-scale advection-diffusion equations because it takes less time to solve than other more expensive VI algorithms like VI - SS. However, if advection becomes negligible thus reducing the system to a symmetric diffusion problem, one could employ either QP -



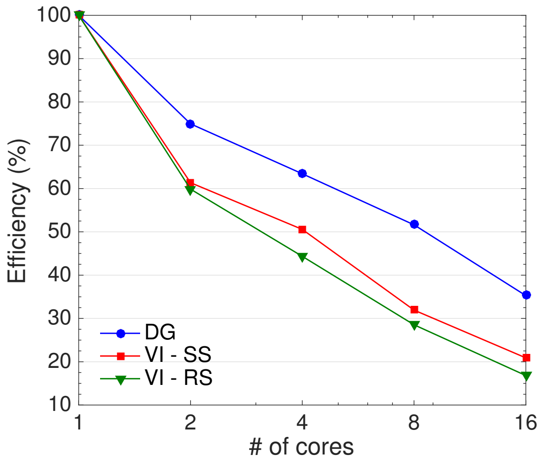
(a) SUPG - solve time



(b) SUPG - parallel efficiency



(c) DG - solve time



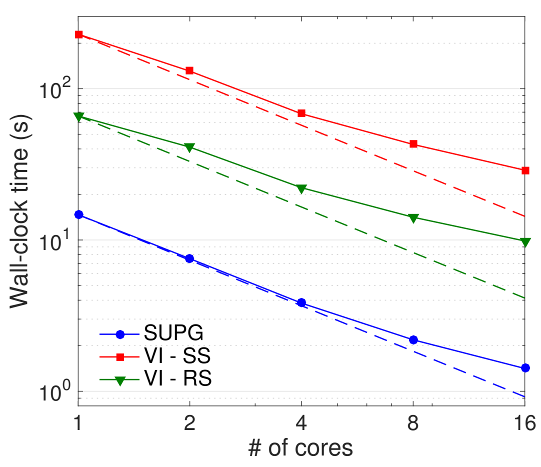
(d) DG - parallel efficiency

Figure 4.15: 3D advection-diffusion: weak-scaling plots with approximately 100k degrees-of-freedom per core and the corresponding parallel efficiencies on a single Intel Xeon E5-2680v2 server.

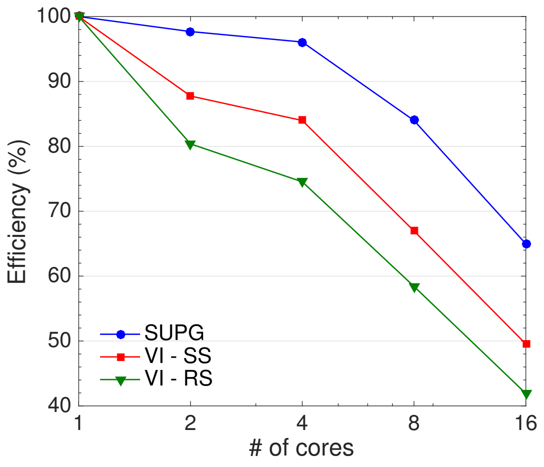
TRON or VI - RS as these solvers are equal in both parallel and algorithmic scalability.

4.5.3 Solver tolerances

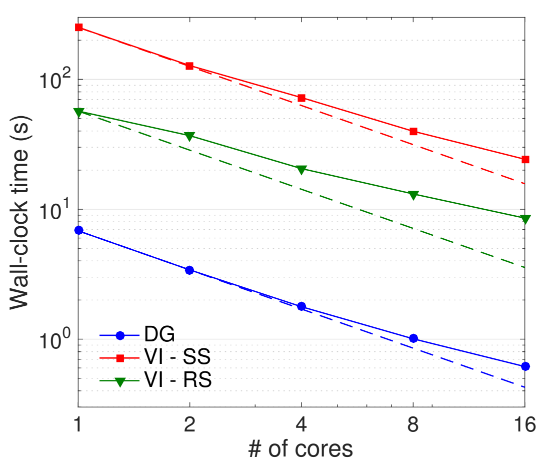
So far the results in this section suggest that the VI approach is fairly computationally expensive for both diffusion and advection-diffusion type equations. This is, partly, because we used a stringent solver convergence criterion (an absolute residual



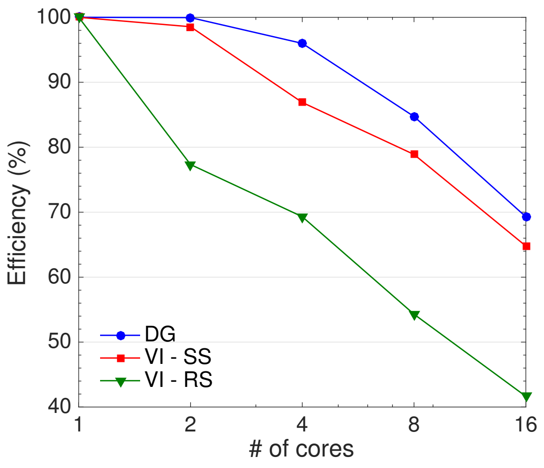
(a) SUPG - solve time



(b) SUPG - parallel efficiency



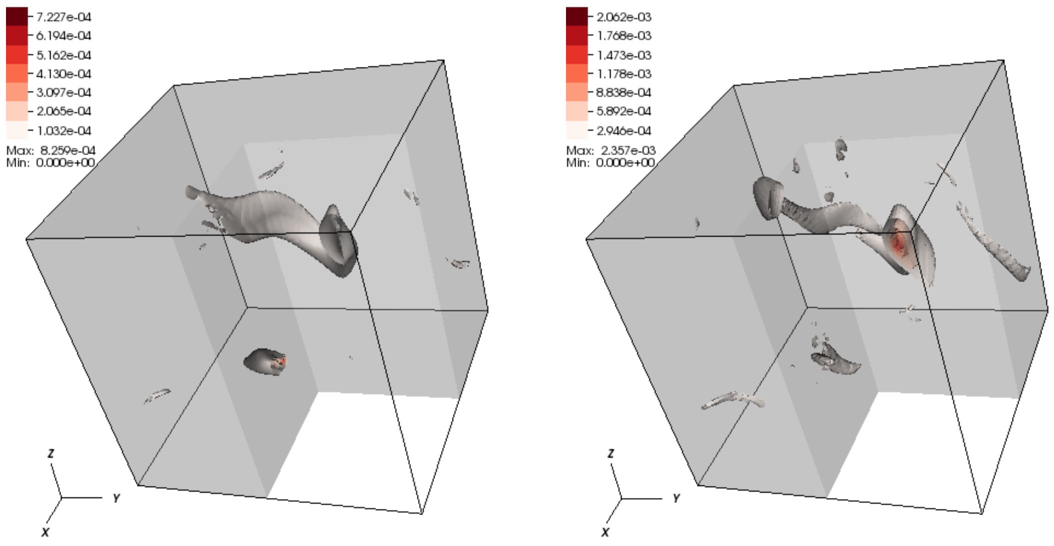
(c) DG - solve time



(d) DG - parallel efficiency

Figure 4.16: 3D advection-diffusion: strong-scaling plots for approximately 500k degrees of freedom (h-size $= 1/80$ and $1/40$ for GAL and DG respectively) and the corresponding parallel efficiencies on a single Intel Xeon E5-2680v2 server.

tolerance of 10^{-8}) for both QP and VI. The exact choice of tolerance is indeed problem and application dependent, but our proposed computational framework sets the solution to the unconstrained problem as the initial guess. Therefore, we have the luxury to use a more relaxed solver convergence criterion and investigate how a relative residual tolerance of 10^{-1} affects the performance of the VI solvers numerically and algorithmically. Table 4.9 contains the time-to-solution and the total number of VI iterations for the advection-diffusion problem under the SUPG and DG formulations.



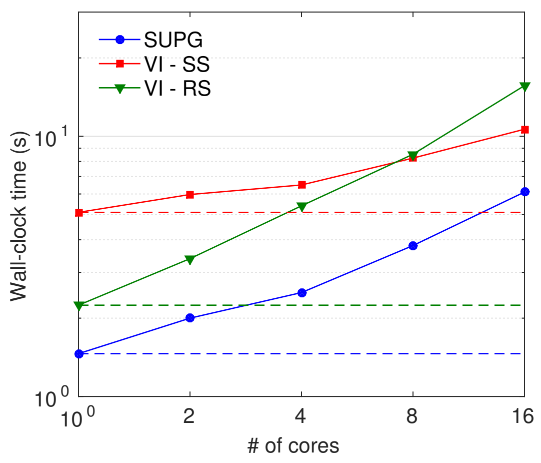
(a) SUPG **(b)** DG

Figure 4.17: 3D advection-diffusion: 3D contours of the absolute difference in VI - RS concentrations between the relative residual tolerance of 10^{-1} solution and the absolute residual tolerance of 10^{-8} solution.

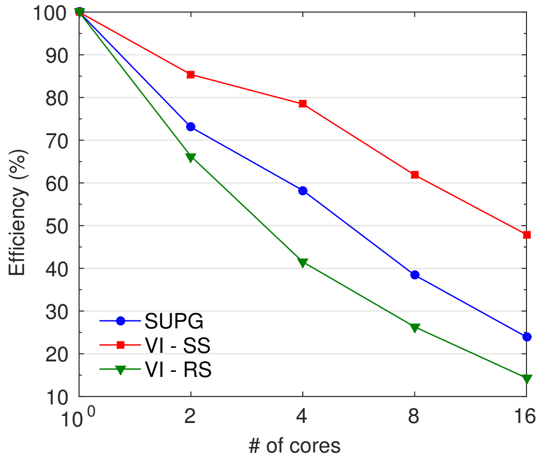
Table 4.9: 3D advection-diffusion: single core wall-clock time on an Intel Xeon E5-2680v2 server and number of VI solve iterations for various levels of mesh refinement when the relative solver tolerance is set to 10^{-1}.

h-size	SUPG: VI - SS		SUPG: VI - RS		DG: VI - SS		DG: VI - RS	
	time (s)	iters	time (s)	iters	time (s)	iters	time (s)	iters
1/10	0.051	2	0.005	1	0.653	2	0.088	2
1/20	0.376	2	0.076	2	5.471	2	1.512	3
1/30	1.316	2	0.315	2	19.64	2	6.693	3
1/40	4.425	3	1.308	3	48.70	2	18.20	3
1/50	6.723	2	3.125	3	97.53	2	50.98	4
1/60	11.80	2	6.232	3	174.2	2	96.47	4
1/70	19.68	2	14.92	4	290.0	2	196.6	5
1/80	30.83	2	24.73	4	430.1	2	305.6	5

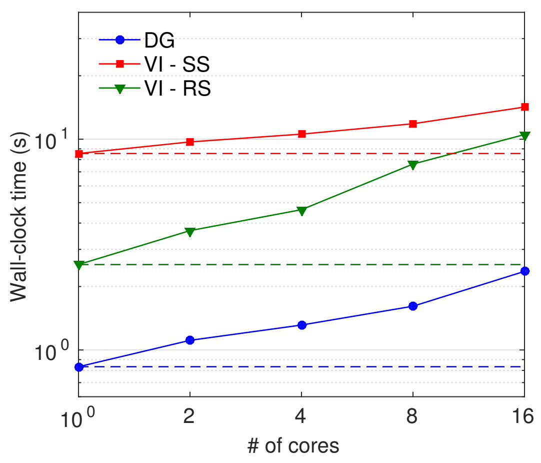
Relaxing the tolerance greatly reduces the amount of wall-clock time and solver iterations needed with acceptable compromise on the accuracy as seen from the absolute difference in concentrations in Figure 4.17. The weak scaling plots in Figures 4.18 indicate that the VI - SS has relatively good algorithmic performance with respect to problem size. whereas the strong-scaling plots in Figure 4.19 still indicate that VI -



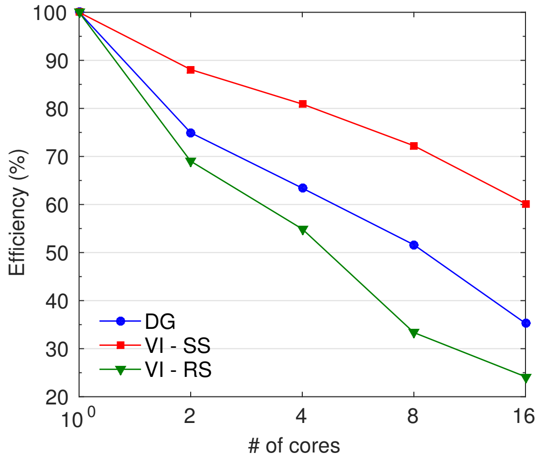
(a) SUPG - solve time



(b) SUPG - parallel efficiency



(c) DG - solve time

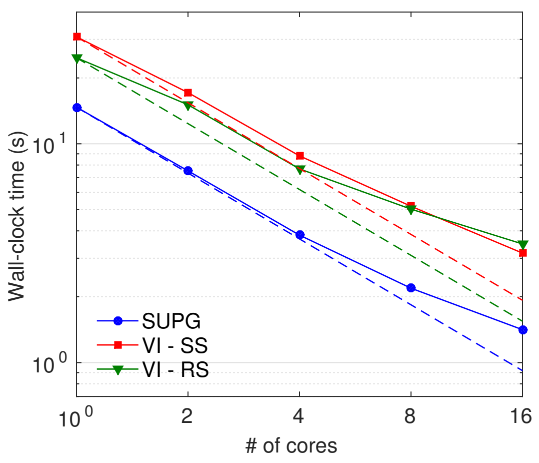


(d) DG - parallel efficiency

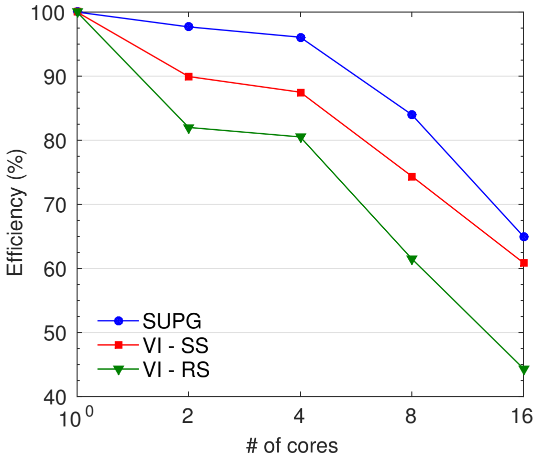
Figure 4.18: 3D advection-diffusion: weak-scaling plots of the relaxed VI solvers with approximately 100k degrees-of-freedom per core and the corresponding parallel efficiencies on a single Intel Xeon E5-2680v2 server.

RS is not as scalable as the VI - SS.

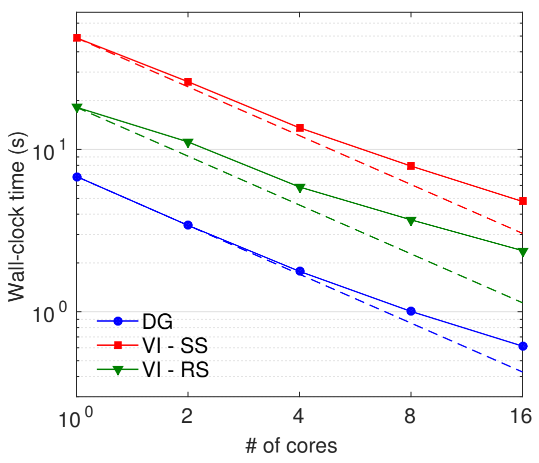
Remark 4.5.1. *These parallel performance studies do not indicate how truly efficient the PETSc and TAO implementations of the QP and VI solvers are in the context of high performance computing. Moreover, strong and weak scaling results can be deceptive as discussed in Chapter 2. It should be noted that a serially efficient algorithm will likely have poor parallel performance due to dominating effects from communication overhead and memory latencies. Also, the use of assembled sparse matrices*



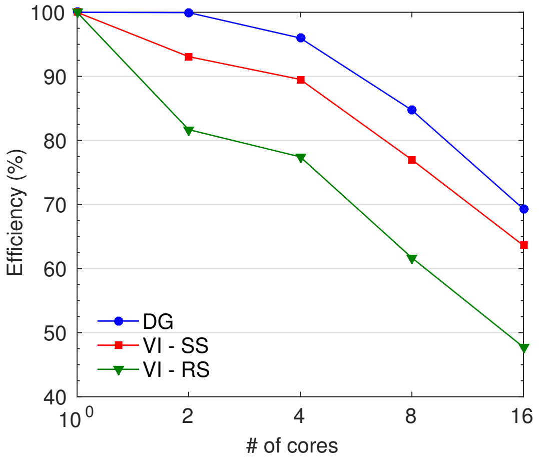
(a) SUPG - solve time



(b) SUPG - parallel efficiency



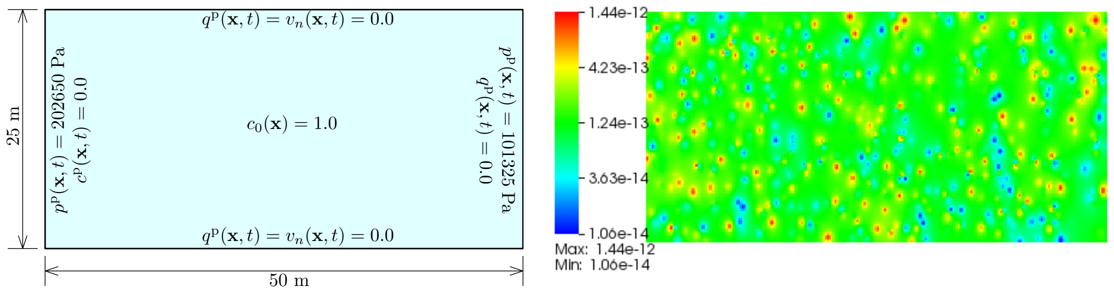
(c) DG - solve time



(d) DG - parallel efficiency

Figure 4.19: 3D advection-diffusion: strong-scaling plots of the relaxed VI solvers for approximately 500k degrees of freedom (h-size = 1/80 and 1/40 for GAL and DG respectively) on a single Intel Xeon E5-2680v2 server.

coupled with memory bandwidth effects explains why the scaling can degrade even when only 2 MPI processes are used. One could use either the performance spectrum model outlined in Chapter 2 or the "perfect cache" roofline model outlined in Chapter 3 to better understand the performance of the VI solvers.


$q^{\mathrm{p}}(\mathbf{x},t) = v_n(\mathbf{x},t) = 0.0$
$p^{\mathrm{p}}(\mathbf{x},t) = 202650$ Pa
$c^{\mathrm{p}}(\mathbf{x},t) = 0.0$
$c_0(\mathbf{x}) = 1.0$
$p^{\mathrm{p}}(\mathbf{x},t) = 101325$ Pa
$q^{\mathrm{p}}(\mathbf{x},t) = 0.0$
$q^{\mathrm{p}}(\mathbf{x},t) = v_n(\mathbf{x},t) = 0.0$
25 m
50 m
1.44e-12
4.23e-13
1.24e-13
3.63e-14
1.06e-14
Max: 1.44e-12
Min: 1.06e-14

(a) Problem description **(b)** Log scale permeability field (m^2)

Figure 4.20: 2D miscible displacement: Pictorial description of the boundary value problems for the coupled Darcy and advection-diffusion equations and the corresponding random permeability.

4.6 EXTENSION TO TRANSIENT ANALYSIS AND COUPLED PROBLEMS

We now illustrate that the proposed computational framework, which is based on variational inequalities, can be extended to perform a transient analysis. The resulting governing equations will then be *parabolic* variational inequalities. This extension will be illustrated by considering the displacement of miscible fluids in porous media wherein a fluid displaces a fluid with higher viscosity [157]. Some of the applications of miscible displacement include oil recovery and carbon-dioxide sequestration [32, 33]. The phenomenon is commonly modeled using coupled flow and transport equations, which will be presented below. In this section, we will also show how negative concentrations can have serious ramifications when simulating non-linear transport phenomenon like the displacement of miscible fluids.

4.6.1 Governing equations and temporal discretization

We denote the time by $t \in [0, \mathcal{T}]$, where $\mathcal{T}$ denotes the length of the time interval of interest. For the Darcy equation, the boundary is divided into two parts: Γ^p and Γ^v, such that $\Gamma^p \cup \Gamma^v = \partial\Omega$ and $\Gamma^p \cap \Gamma^v = \emptyset$. Γ^p and Γ^v denote the parts of the boundary on which pressure and velocity boundary conditions are enforced respectively. We shall denote time-dependent pressure by $p(\mathbf{x}, t)$, time-dependent velocity by $\mathbf{v}(\mathbf{x}, t)$, concentration-dependent viscosity by $\mu(c(\mathbf{x}, t))$, permeability by $k(\mathbf{x})$, density by ρ, time-dependent specific body force by $\mathbf{b}(\mathbf{x}, t)$, time-dependent concentration by $c(\mathbf{x}, t)$, prescribed initial concentration by $c_0(\mathbf{x})$, time dependent volumetric source by $f(\mathbf{x}, t)$, and time-dependent diffusivity tensor by $\mathbf{D}(\mathbf{x}, t)$. For the boundary conditions, the prescribed time-dependent concentration is denoted by $c^{\mathrm{p}}(\mathbf{x}, t)$, prescribed time-dependent pressure by $p^{\mathrm{p}}(\mathbf{x}, t)$, prescribed time-dependent normal component of the velocity by $v_n(\mathbf{x}, t)$, and prescribed time-dependent flux by $q^{\mathrm{p}}(\mathbf{x}, t)$. The initial boundary value problem for the coupled flow and advective-diffusive equations can be written as follows:

$$\frac{\mu(c(\mathbf{x}, t))}{k(\mathbf{x})}\mathbf{v}(\mathbf{x}, t) + \mathrm{grad}[p(\mathrm{x}, t)] = \rho\mathbf{b}(\mathbf{x}, t) \quad \text{in } \Omega \times (0, \mathcal{T}), \tag{4.6.1a}$$

$$\mathrm{div}[\mathbf{v}(\mathrm{x}, t)] = 0 \quad \text{in } \Omega \times (0, \mathcal{T}), \tag{4.6.1b}$$

$$p(\mathbf{x}, t) = p^{\mathrm{p}}(\mathbf{x}, t) \quad \text{on } \Gamma^{p} \times (0, \mathcal{T}), \tag{4.6.1c}$$

$$\mathbf{v}(\mathbf{x}, t) \cdot \widehat{\mathbf{n}} = v_n(\mathbf{x}, t) \quad \text{on } \Gamma^{v} \times (0, \mathcal{T}), \tag{4.6.1d}$$

$$\frac{\partial c(\mathbf{x}, t)}{\partial t} + \mathbf{v}(\mathbf{x}, t) \cdot \mathrm{grad}[c(\mathbf{x}, t)] - \mathrm{div}[\mathbf{D}(\mathbf{x}, t)\mathrm{grad}[c(\mathbf{x}, t)]] = f(\mathbf{x}, t) \quad \text{in } \Omega \times (0, \mathcal{T}), \tag{4.6.1e}$$

$$c(\mathbf{x}, t) = c^{\mathrm{p}}(\mathbf{x}, t) \quad \text{on } \Gamma^{\mathrm{D}} \times (0, \mathcal{T}), \tag{4.6.1f}$$

$$\widehat{\mathbf{n}}(\mathbf{x}) \cdot (\mathbf{v}(\mathbf{x}, t)c(\mathbf{x}, t) - \mathbf{D}(\mathbf{x}, t)\mathrm{grad}[c(\mathbf{x}, t)]) = q^{\mathrm{p}}(\mathbf{x}, t) \quad \text{on } \Gamma^{\mathrm{N}}_{\text{inflow}} \times (0, \mathcal{T}), \tag{4.6.1g}$$

$$-\widehat{\mathbf{n}}(\mathbf{x}) \cdot \mathbf{D}(\mathbf{x}, t)\mathrm{grad}[c(\mathbf{x}, t)] = q^{\mathrm{p}}(\mathbf{x}, t) \quad \text{on } \Gamma^{\mathrm{N}}_{\text{outflow}} \times (0, \mathcal{T}), \text{ and} \tag{4.6.1h}$$

$$c(\mathbf{x}, 0) = c_0(\mathbf{x}) \quad \text{in } \Omega, \tag{4.6.1i}$$

where equations (4.6.1a) through (4.6.1d) represent the Darcy equation, and equations (4.6.1e) through (4.6.1i) represent the transient advection-diffusion equation. To complete the coupled problem, the viscosity is assumed to depend exponentially

Table 4.10: 2D miscible displacement: problem parameters

Parameter	Value
$\mathbf{b}(\boldsymbol{x})$	$\{0, 0\}$ m/s^2
$\mu(c(\mathbf{x}, t))$	See equation (4.6.2a)
μ_0	$3.95 \cdot 10^{-5}$ Pa s
R_c	3
$k(\mathbf{x})$	varies
$f(\mathbf{x}, t)$	0
ρ	479 kg/m^3
α_L	10^{-1} m
α_T	10^{-5} m
α_D	10^{-9} m^2/s
Number of elements	31,250
Darcy degrees-of-freedom	94,125
Advection-diffusion degrees-of-freedom	125,000

on concentration:

$$\mu(c(\mathbf{x}, t)) = \mu_0 \exp\Big[R_c c(\mathbf{x}, t)\Big] \quad \text{and} \tag{4.6.2a}$$

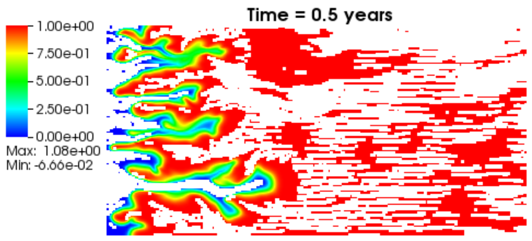
$$\mu(c(\mathbf{x}, t)) = \mu_0 \exp\Big[R_c(1 - c(\mathbf{x}, t))\Big], \tag{4.6.2b}$$

where μ_0 is the base viscosity of the less viscous fluid and R_c is the log-mobility ratio in an isothermal miscible displacement. To solve the transient advection-diffusion equation, we employ the method of horizontal lines [66], which first discretizes the time derivatives, thereby giving rise to time-independent equations. The time interval of interest is divided into N sub-intervals. That is,

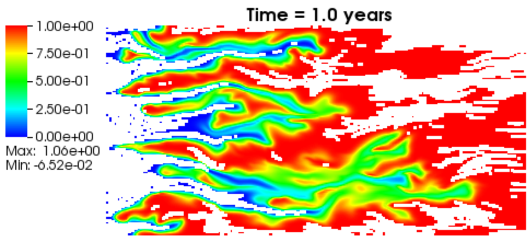
$$[0, \mathcal{T}] := \bigcup_{n=0}^{N} [t_n, t_{n+1}], \tag{4.6.3}$$

where t_n denotes the n-th time-level. We assume that the time-step is uniform, which can be written as:

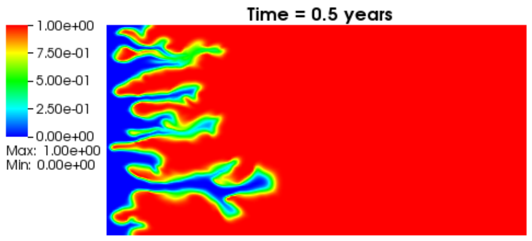
$$\Delta t = t_{n+1} - t_n. \tag{4.6.4}$$



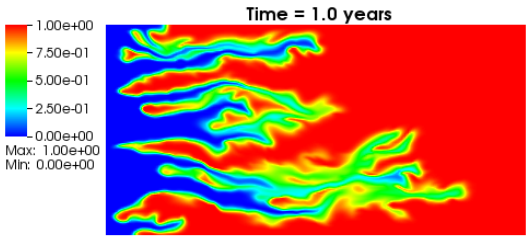
(a) $\boldsymbol{c}_{\mathrm{DG}}$ at $t = 0.5$ years



(b) $\boldsymbol{c}_{\mathrm{DG}}$ at $t = 1.0$ years



(c) $\boldsymbol{c}_{\mathrm{RS}}$ at $t = 0.5$ years



(d) $\boldsymbol{c}_{\mathrm{RS}}$ at $t = 1.0$ years

Figure 4.21: 2D miscible displacement: Concentration fields under the DG ($\boldsymbol{c}_{\mathrm{DG}}$) formulation and VI - RS ($\boldsymbol{c}_{\mathrm{RS}}$) method at various time levels. White regions denote violations of the maximum principle and the non-negative constraint.

One can then employ the finite-dimensional VI solvers for these resulting equations, which were described earlier in this chapter. This implies that we will still be solving elliptic VIs of first kind but at each time level. This procedure will be illustrated below using the backward Euler method. However, a detailed discussion on the effect of time-stepping schemes in meeting maximum principles can be found in [131]. For a transient analysis, the proposed framework outlined in Section 4.4.2 is modified as follows:

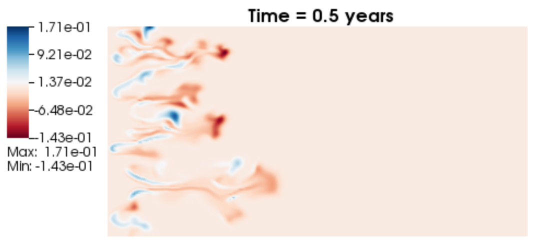
1. Set $t = 0.0$, $n = 0$, and $\boldsymbol{c}^{(n)} = \boldsymbol{c}_0$.

2. Solve Darcy equation:

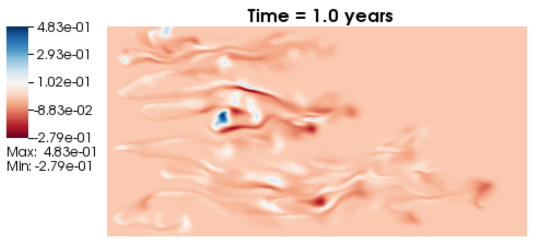
   (a) Compute $\mu(\boldsymbol{c}^{(n)})$ using equation (4.6.2).

   (b) Assemble $\boldsymbol{K}_{vv}$, $\boldsymbol{K}_{vp}$, $\boldsymbol{K}_{pv}$, $\boldsymbol{K}_{pp}$, $\boldsymbol{f}_v$ and $\boldsymbol{f}_p$.

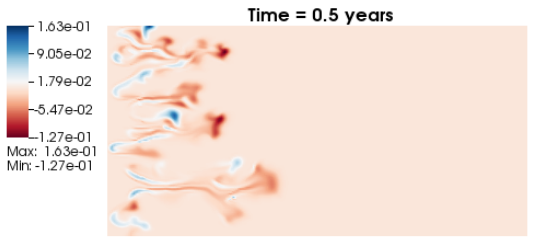
   (c) Solve for $\boldsymbol{v}^{(n)}$.



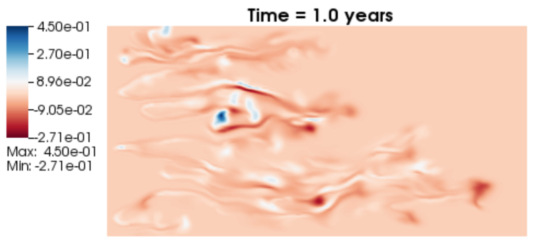
(a) $\boldsymbol{c}_{\text{CLIP}}$ - $\boldsymbol{c}_{\text{RS}}$ at $t = 0.5$ years



(b) $\boldsymbol{c}_{\text{CLIP}}$ - $\boldsymbol{c}_{\text{RS}}$ at $t = 1.0$ years



(c) $\boldsymbol{c}_{\text{DG}}$ - $\boldsymbol{c}_{\text{RS}}$ at $t = 0.5$ years



(d) $\boldsymbol{c}_{\text{DG}}$ - $\boldsymbol{c}_{\text{RS}}$ at $t = 1.0$ years

Figure 4.22: 2D miscible displacement: Differences between concentration fields under the DG formulation ($\boldsymbol{c}_{\text{DG}}$), VI - RS method ($\boldsymbol{c}_{\text{RS}}$), and clipping procedure ($\boldsymbol{c}_{\text{CLIP}}$).

3. Solve advection-diffusion equation:

   (a) Compute $\mathbf{D}^{(n)}$ using equation (4.5.3).

   (b) Assemble $\boldsymbol{K}_c$ and $\boldsymbol{f}_c$ using $\boldsymbol{c}^{(n)}$.

   (c) Solve for $\boldsymbol{c}_{\text{DG}}^{(n+1)}$.

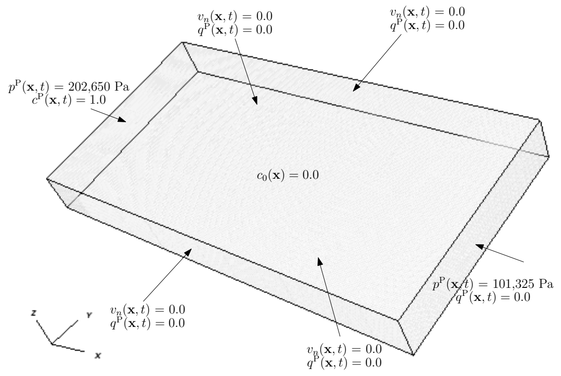
   (d) Clip $\boldsymbol{c}_{\text{DG}}^{(n+1)}$ and obtain $\boldsymbol{c}_{\text{CLIP}}^{(n+1)}$.

   (e) Solve the bounded constraint problem for $\boldsymbol{c}_{\text{RS}}^{(n+1)}$ with $\boldsymbol{c}_{\text{CLIP}}^{(n+1)}$ as the initial guess.

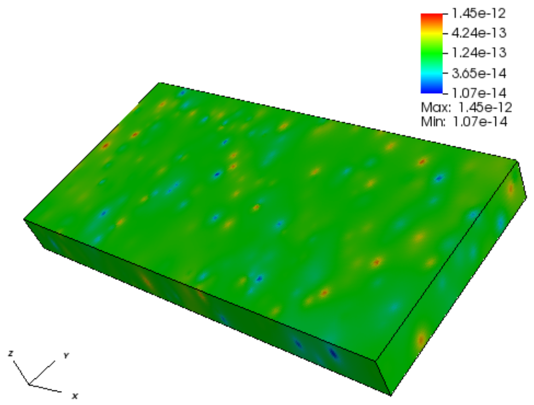
4. Set $\boldsymbol{c}^{(n+1)} \longleftarrow \boldsymbol{c}_{\text{RS}}^{(n+1)}$, $t \longleftarrow t + \Delta t$, and $n \longleftarrow n + 1$.

5. If $n < N$ go to Step2.

where $\boldsymbol{K}_{vv}$, $\boldsymbol{K}_{vp}$, $\boldsymbol{K}_{pv}$, $\boldsymbol{K}_{pp}$, $\boldsymbol{f}_v$, and $\boldsymbol{f}_p$ are the assembled matrices and vectors for the Darcy equation, and $\boldsymbol{K}_c$ and $\boldsymbol{f}_c$ are for the transient advection-diffusion equation.


$v_n(\mathbf{x},t) = 0.0$
$q^{\mathrm{P}}(\mathbf{x},t) = 0.0$
$v_n(\mathbf{x},t) = 0.0$
$q^{\mathrm{P}}(\mathbf{x},t) = 0.0$
$p^{\mathrm{P}}(\mathbf{x},t) = 202{,}650$ Pa
$c^{\mathrm{P}}(\mathbf{x},t) = 1.0$
$c_0(\mathbf{x}) = 0.0$
$p^{\mathrm{P}}(\mathbf{x},t) = 101{,}325$ Pa
$q^{\mathrm{P}}(\mathbf{x},t) = 0.0$
$v_n(\mathbf{x},t) = 0.0$
$q^{\mathrm{P}}(\mathbf{x},t) = 0.0$
$v_n(\mathbf{x},t) = 0.0$
$q^{\mathrm{P}}(\mathbf{x},t) = 0.0$
z
Y
x

(a) Problem description



(b) Log scale permeability field (m^2)

Figure 4.23: 3D miscible displacement: Pictorial description of the boundary value problems (50m×25m×5m domain) for the coupled Darcy and advection-diffusion equations and the corresponding random permeability.

The finite element discretization and solution strategy for the steady-state Darcy equations can be found in Appendix B.

4.6.2 Numerical results

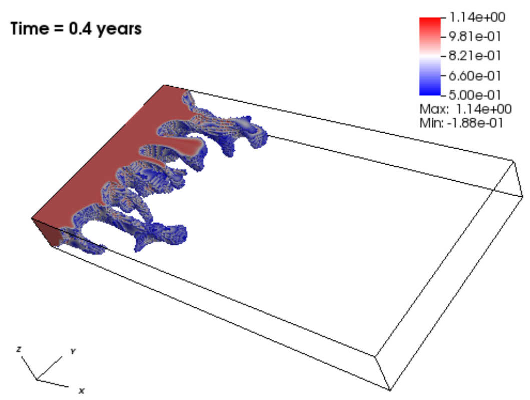
Consider a 50m×25m rectangular domain with heterogeneous permeability, as shown in Figure 4.20. The flow will be modeled using Darcy equations, in which the viscosity depends on the concentration of the attendant chemical species, and the transport of the chemical species will be modeled using advection-diffusion equations. For the flow subproblem, we prescribe the pressure boundary conditions on the left and right sides of the domain and no flow boundary conditions on the top and bottom. For the transport subproblem, an initial concentration of unity is prescribed everywhere in the domain, and a Dirichlet boundary condition of zero along the left side and zero flux boundary conditions on the remaining sides are prescribed. A time-step $\Delta t = 1$ day is used to simulate the miscible displacement over a time interval $\mathcal{T} = 1$ year. All other problem parameters and material properties can be found in Table 4.10. Figure 4.21 depicts the concentration profiles under the DG and

Table 4.11: 3D miscible displacement: problem parameters

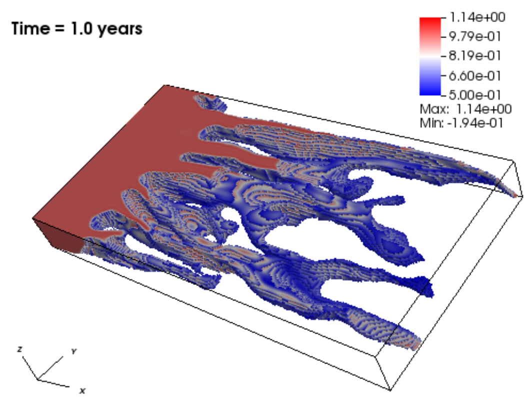
Parameter	Value
$\mathbf{b}(\boldsymbol{x})$	$\{0, 0, -9.81\}$ m/s^2
$\mu(c(\mathbf{x}, t))$	See equation (4.6.2b)
μ_0	$3.95 \cdot 10^{-5}$ Pa s
R_c	3
$k(\mathbf{x})$	varies
$f(\mathbf{x}, t)$	0
ρ	479 kg/m^3
α_L	10^{-1} m
α_T	10^{-5} m
α_D	10^{-9} m^2/s
Number of elements	781,250
Darcy degrees-of-freedom	3,165,625
Advection-diffusion degrees-of-freedom	6,250,000

VI - RS methods at $t = 0.5$ and $t = 1.0$ years. It can be seen that violations in the maximum principles occur even under the coupled flow and transport computational framework. Furthermore, the violations do not go away as the simulation progresses in time. As it may be difficult to distinguish between the VI - RS and DG or clipping procedures by directly plotting the solutions, we show the differences in the solutions in Figure 4.22. The ramification shown by the difference plots is that the development of the plumes and fingerings throughout the computational domain is altered by the VI approach. It is also shown that the clipping procedure is significantly different from the VI approach.

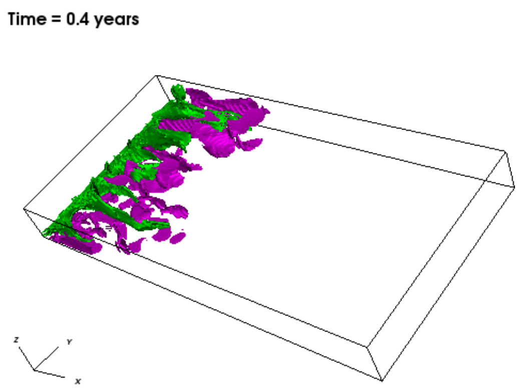
To understand the performance of our VI-based solvers for larger versions of this problem, we now consider a 50m$\times$25m$\times$5m box domain with heterogeneous permeability as shown in Figure 4.23. Table 4.11 lists all the necessary problem parameters, and the same time-step and time interval from the previous problem is used. This problem is now solved in parallel across 40 MPI processes with a relative residual convergence criterion of 10^{-3}, and Figure 4.24 depicts the numerical results under the DG formulation without VI - RS and DG formulation with VI - RS. The exact regions where violations in the maximum principle and the non-negative constraint occur are



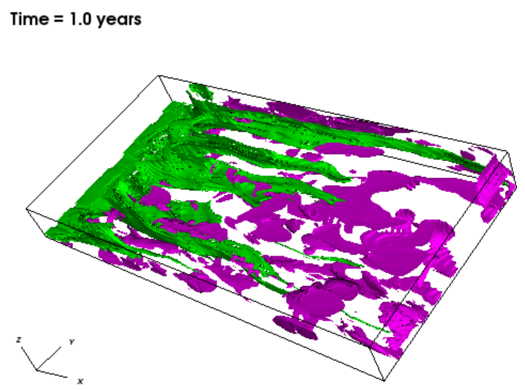
(a) c_{DG} at $t = 0.4$ years



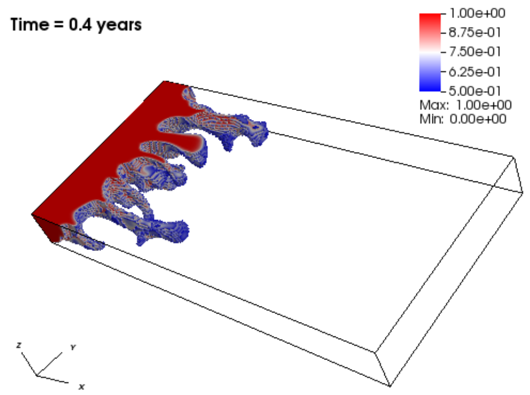
(b) c_{DG} at $t = 1.0$ years



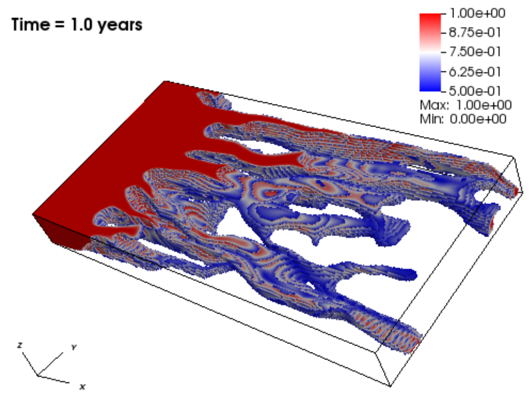
(c) c_{DG} at $t = 0.4$ years



(d) c_{DG} at $t = 1.0$ years



(e) c_{RS} at $t = 0.4$ years



(f) c_{RS} at $t = 1.0$ years

Figure 4.24: 3D miscible displacement: Top (DG) and bottom (VI) show regions with concentrations above 0.5. Middle figures show regions with concentrations above 1.0 (green) and below 0.0 (purple).

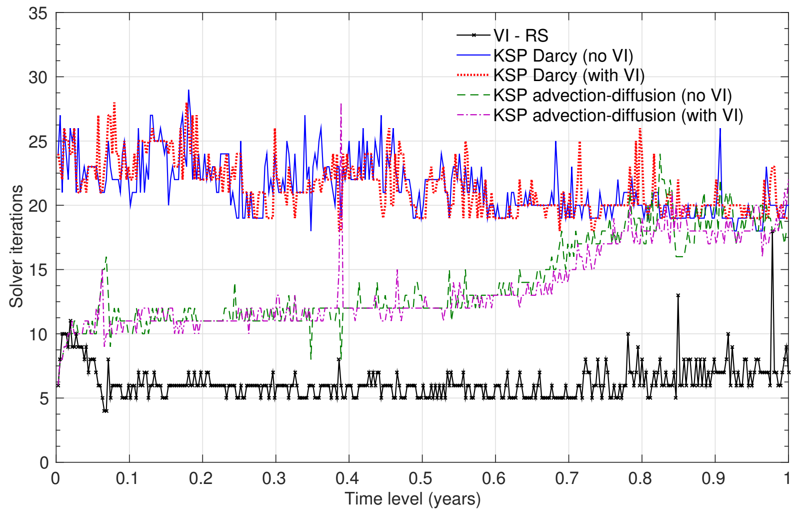


Figure 4.25: 3D miscible displacement: Total number of KSP iterations at each time level for the Darcy and advection-diffusion equation with and without VI. Also shown is the total number of VI iterations at each time level.

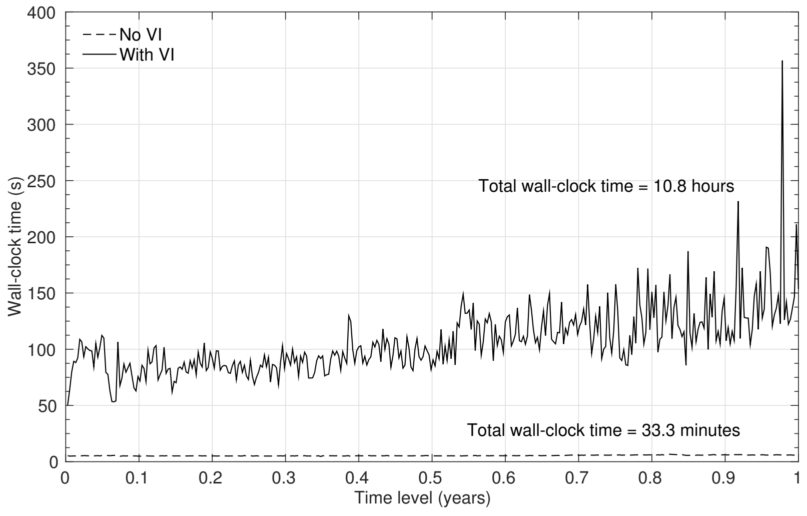


Figure 4.26: 3D miscible displacement: The wall-clock time as a function of simulated time with and without VI across 40 cores (two Intel Xeon E5-2680v2 servers). Also shown is the sum of the wall-clock time across all time levels.

shown in Figures 4.24c and 4.24d. First, we note that this proposed computational framework can successfully eliminate the violations that occur in a large-scale miscible displacement simulation. We also note that the development and displacement of the plumes is significantly affected by whether VI - RS is applied or not; the differences between Figures 4.24b and 4.24f are quite evident unlike the 2D example. Next, we note from Figure 4.25 that enforcing the bounded constraints under the VI - RS method will not drastically increase the total number of KSP iterations needed for either the Darcy or advection-diffusion equations. However, the wall-clock time shown in Figure 4.26 indicates that the VI - RS method is very time consuming. The oscillatory behavior of both the solver iterations and wall-clock time at each time level is largely attributed to the heterogeneous nature of the problem as well as the number of maximum principle violating degrees-of-freedom that naturally arise out of the DG formulation. Although VI-based solvers like VI - RS can enforce maximum principles and the non-negative constraint, we have observed that applying such methodologies can make the overall advection-diffusion finite element simulation up to 20 times as expensive even in a parallel environment.

Before we draw any further conclusions of this chapter, we acknowledge that we did not perform a numerical *h*-convergence study. This is due to, despite our best efforts, failure to find an advection-diffusion boundary value problem that considers anisotropy, has an analytical solution, and violates discrete maximum principles. We, therefore, illustrated the performance of the proposed computational framework through other means, as presented in the previous sections.

4.7 CONCLUDING REMARKS

We presented a robust computational framework based on VIs for diffusion and advection-diffusion equations that satisfies the discrete maximum principles and the non-negative constraint. The framework is applicable to *large-scale* and transient problems, and can be solved in a parallel setting. The main contributions of this chapter and the salient features of the proposed formulation can be summarized as follows.

1. Realizing and posing the advection-diffusion problem as a variational inequality (VI) to meet the discrete maximum principles and the non-negative constraint.

2. For large-scale problems, we have demonstrated that QP solvers, which is a special case of VIs, are just as good as VI solvers for symmetric and positive-definite problems like the diffusion equation. On the other hand, the proposed VI-based framework can also handle non-self-adjoint operators.

3. Unlike the non-negative framework proposed in [120], which is based on a mixed least-squares WF, the proposed framework can utilize any finite element formulation including single-field formulations, and these formulations need not result in symmetric and positive definite coefficient matrices.

4. The proposed framework allows one to leverage on existing state-of-the-art computational frameworks for solving VIs. In particular, the Firedrake project, which provides access to parallel solvers in PETSc and TAO libraries, can serve as a suitable platform for implementing the proposed framework, as illustrated in this chapter.

5. This framework is suitable for many important applications like miscible displacement, subsurface remediation, and transport of radionuclides. In these applications, one encounters not only highly anisotropic medium properties but also highly non-linear phenomena due to aqueous complexation and kinetic reactions.

Chapter 5. Conclusions

Subsurface flow and transport modeling is highly critical for energy related applications, so it is important to not only have the software packages and algorithms needed to enhance current predictive capabilities but also a thorough understanding of their computational performances when large-scale problems need to be addressed. We have presented a high performing parallel computational framework that ensures element-wise/local mass conservation through the Discontinuous Galerkin finite element method, enforces the non-negative constraint through the variational inequality approach, and is able to solve relevant subsurface transport problems in a parallel setting. The performance of the software and tools used for this framework is justified using concepts deriving from a newly proposed performance spectrum model. The main contributions of this dissertation can be summarized as follows:

1. First, we have designed a conceptual *performance spectrum* model which simultaneously illuminates the impacy of both hardware and algorithmic efficiencies. As proof-of-concept, we displayed its unique capabilities to compare various approaches to numerical simulation of PDE's across hardware platforms, software implementations, algorithms, and numerical discretization.

2. Second, we have outlined a PETSc-based parallel computational framework that employs optimization-based routines to enforce the non-negative constraint for diffusion equations. The performance model presented has shown that the framework is scalable and is suitable for large-scale subsurface problems that the DOE face every day.

3. Third, we have extended the PETSc-based framework to enforce the non-negative constraint for advection-diffusion equations by using the variational

inequality approach. Its robustness was displayed by solving the miscible displacement of oil in a field-scale heterogeneous reservoir.

5.1 FUTURE WORK

Based on the research conducted in this dissertation, we shall outline four promising future and ongoing research tasks. On the *high performance computing* front, the following issues can be studied:

- Most of the HPC systems chosen for this dissertation are from Intel and AMD and have similar performance characteristics, but processors from other vendors such as IBM's POWER8 [155] or ARM-based systems [140] may tell a different story. HPC architecture is constantly evolving, and simple benchmarks and measurements like STREAM Triad may not be sufficient for understanding the performance of complex solvers or algorithms on these state-of-the-art machines.
- The intensity equations presented in Section 2.3 are relatively easy to incorporate into any code, but they only provide relative comparisons between various flavors of hardware and software. An improved methodology for documenting the [Work] and [TBT] metrics (which represent the total floating-point operations and total bytes transferred, respectively, as described in Chapter 2) would certain improve the predictive capabilities of the performance spectrum model.

On the *numerical* front, the following issues can be studied:

- For subsurface flow models, pressure arising from Darcy's model cannot be negative nor violate maximum principles. Multi-physics problems require special preconditioning and solution methodologies, so a possible research endeavor is to couple the variational inequality approach with PETSc's block solvers to ensure physically realistic pressures. Non-linear flow through porous media models such as Richard's equation and multi-phase flows also need to be addressed as these equations are commonly needed for subsurface and energy applications.

- Finally, the proposed computational framework can be extended to advective-diffusive-reactive geochemical systems. These equations are highly non-linear and are notoriously difficult to solve efficiently. Negative concentrations may arise not only because of non-monotone numerical methods but also because of Newton's method potentially converging to negative roots in the concentration solution.

References

[1] Adams, M. F., Bayraktar, H. H., Keaveny, T. M., and Papadopoulos, P. (2004). "Ultrascalable Implicit Finite Element Analyses in Solid Mechanics with Over a Half a Billion Degrees of Freedom." *ACM/IEEE Proceedings of SC2004: High Performance Networking and Computing.* Gordon Bell Award.

[2] Adams, R. J. and Fournier, J. J. F. (2003). *Sobolev Spaces*, Vol. 140. Academic press.

[3] Adhianto, L., Banerjee, S., Fagan, M., Krentel, M., Marin, G., Mellor-Crummey, J., and Tallent, N. R. (2010). "HPCToolkit: Tools for Performance Analysis of Optimized Parallel Programs." *Concurrency and Computation: Practice and Experience*, 22(6), 685–701.

[4] Alnæs, M., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M. E., and Wells, G. N. (2015). "The FEniCS Project Version 1.5." *Archive of Numerical Software*, 3(100), 9–23.

[5] Alnæs, M. S., Logg, A., Ølgaard, K. B., Rognes, M. E., and Wells, G. N. (2014). "Unified Form Language: A Domain-specific Language for Weak Formulations of Partial Differential Equations." *ACM Transactions on Mathematical Software*, 40(2), 9.

[6] Amdahl, G. M. (1967). "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities." *Proceedings of the April 18-20, 1967, spring joint computer conference*, ACM. 483–485.

[7] Amestoy, P. R., Duff, I. S., L'Excellent, J., and Koster, J. (2000). "MUMPS: A General Purpose Distributed Memory Sparse Solver." *International Workshop on Applied Parallel Computing*, Springer. 121–130.

[8] Armijo, L. (1966). “Minimization of Functions Having Lipschitz-Continuous First Partial Derivatives.” *Pacific Journal of Mathematics*, 16, 1–3.

[9] Arnold, D. N., Brezzi, F., Cockburn, B., and Marini, L. D. (2002). “Unified Analysis of Discontinuous Galerkin Methods for Elliptic Problems.” *SIAM Journal on Numerical Analysis*, 39, 1749–1779.

[10] Ayachit, U. (2015). *The ParaView Guide: A Parallel Visualization Application*. Kitware.

[11] Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Rupp, K., Smith, B. F., Zampini, S., Zhang, H., and Zhang, H. (2014). “PETSc Web Page. http://www.mcs.anl.gov/petsc.

[12] Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Rupp, K., Smith, B. F., Zampini, S., Zhang, H., and Zhang, H. (2016). “PETSc Users Manual.” *Report No. ANL-95/11 - Revision 3.7*, Argonne National Laboratory.

[13] Balay, S., Gropp, W. D., McInnes, L. C., and Smith, B. F. (1997). “Efficient Management of Parallelism in Object Oriented Numerical Software Libraries.” *Modern Software Tools in Scientific Computing*, E. Arge, A. M. Bruaset, and H. P. Langtangen, eds. Birkhäuser Press, 163–202.

[14] Bangerth, W., Hartmann, R., and Kanschat, G. (2007). “DEAL.II – A General Purpose Object Oriented Finite Element Library.” *ACM Trans. Math. Softw.*, 33(4), 24/1–24/27.

[15] Benson, S. and Munson, T. S. (2006). “Flexible Complementarity Solvers for Large-Scale Applications.” *Pacific Journal of Mathematics*, 21, 155–168.

[16] Benzi, M., Golub, G. H., and Liesen, J. (2005). "Numerical Solution of Saddle Point Problems." *Acta Numerica*, 14, 1–137.

[17] Bercea, G. T., McRae, A. T. T., Ham, D. A., Mitchell, L., Rathgeber, F., Nardi, L., Luporini, F., and Kelly, P. H. J. (2016). "A Structure-Exploiting Numbering Algorithm for Finite Elements on Extruded Meshes, and its Performance Evaluation in Firedrake." *Geoscientific Model Development*, 9, 3803–3815.

[18] Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization.* Cambridge University Press, Cambridge, UK.

[19] Bramble, J. H., Kwak, D. Y., and Pasciak, J. E. (1994). "Uniform Convergence of Multigrid V-Cycle Iterations for Indefinite and Nonsymmetric Problems." *SIAM Journal on Numerical Analysis*, 31(6), 1746–1763.

[20] Brandt, A. and Livne, O. (2011). *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics, Revised Edition.* SIAM.

[21] Brenner, S. C. and Scott, L. R. (1994). *The Mathematical Theory of Finite Element Methods.* Springer-Verlag, New York, USA.

[22] Brezis, H. (2010). *Functional Analysis, Sobolev Spaces and Partial Differential Equations.* Springer Science & Business Media.

[23] Brooks, A. N. and Hughes, T. J. R. (1982). "Streamline Upwind/Petrov-Galerkin Formulations for Convection Dominated Flows with Particular Emphasis on the Incompressible Navier-Stokes Equations." *Computer Methods in Applied Mechanics and Engineering*, 32, 199–259.

[24] Brown, J. (2016). private communication.

[25] Brown, J., Knepley, M. G., May, D. A., McInnes, L. C., and Smith, B. F. (2012). "Composable Linear Solvers for Multiphysics." *Proceeedings of the 11th*

International Symposium on Parallel and Distributed Computing (ISPDC 2012). IEEE Computer Society, 55–62.

[26] Brown, J., Smith, B., and Ahmadia, A. (2013). “Achieving Textbook Multigrid Efficiency for Hydrostatic Ice Sheet Flow.” *SIAM Journal on Scientific Computing*, 35, B359–B375.

[27] Brune, P. R., Knepley, M. G., Smith, B. F., and Tu, X. (2015). “Composing Scalable Nonlinear Algebraic Solvers.” *SIAM Review*, 57(4), 535–565. `http://www.mcs.anl.gov/papers/P2010-0112.pdf`.

[28] Burdakov, O., Kapyrin, I., and Vassilevski, Y. (2012). “Monotonicity Recovering and Accuracy Preserving Optimization Methods for Postprocessing Finite Element Solutions.” *Journal of Computational Physics*, 231, 3126–3142.

[29] Castelletto, N., White, J. A., and Ferronato, M. (2016). “Scalable Algorithms for Three-Field Mixed Finite Element Coupled Poromechanics.” *Journal of Computational Physics*, 327, 894 – 918.

[30] Chang, J., Karra, S., and Nakshatrala, K. B. (2017). “Large-scale Optimization-Based Non-Negative Computational Framework for Diffusion Equations: Parallel Implementation and Performance Studies.” *Journal of Scientific Computing*, 70, 243–271.

[31] Chang, J. and Nakshatrala, K. B. (2017). “Variational Inequality Approach to Enforce the Non-Negative Constraint for Advection-Diffusion Equations.” *Computer Methods in Applied Mechanics and Engineering*, 320, 287–334.

[32] Chen, C. Y. and Meiburg, E. (1998a). “Miscible Porous Media Displacements in the Quarter Five-Spot Configuration. Part 1. The Homogeneous Case.” *Journal of Fluid Mechanics*, 371, 233–268.

[33] Chen, C. Y. and Meiburg, E. (1998b). “Miscible Porous Media Displacements in the Quarter Five-Spot Configuration. Part 2. Effect of Heterogeneities.” *Journal of Fluid Mechanics*, 371, 269–299.

[34] Chevalier, C. and Pellegrini, F. (2008). “PT-Scotch: A Tool for Efficient Parallel Graph Ordering.” *Parallel computing*, 34(6), 318–331.

[35] Childs, H., Brugger, E., Whitlock, B., Meredith, J., Ahern, S., Pugmire, D., Biagas, K., Miller, M., Harrison, C., Weber, G. H., Krishnan, H., Fogal, T., Sanderson, A., Garth, C., Bethel, E. W., Camp, D., Rübel, O., Durant, M., Favre, J. M., and Navrátil, P. (2012). “VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data.” *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*, 357–372.

[36] Chipot, M. (2009). *Elliptic Equations: An Introductory Course.* Birkhäuser, Basel, Switzerland.

[37] Ciarlet, P. G. and Raviart, P. A. (1973). “Maximum Principle and Uniform Convergence for the Finite Element Method.” *Computer Methods in Applied Methods and Engineering*, 2, 17–31.

[38] Cockburn, B. (2003). “Discontinuous Galerkin Methods.” *Zeitschrift für Angewandte Mathematik und Mechanik*, 83(11), 731–754.

[39] Cockburn, B., Gopalakrishnan, J., and Lazarov, R. (2009). “Unified Hybridization of Discontinuous Galerkin, Mixed, and Continuous Galerkin Methods for Second Order Elliptic Problems.” *SIAM Journal on Numerical Analysis*, 47(2), 1319–1365.

[40] da Veiga, L. B., Lipnikov, K., and Manzini, G. (2014). *The Mimetic Finite Difference Method for Elliptic Problems*, Vol. 11. Springer.

[41] Dagum, L. and Menon, R. (1998). "OpenMP: An Industry Standard API for Shared-Memory Programming." *IEEE computational science and engineering*, 5(1), 46–55.

[42] Dalcin, L. D., Paz, R. R., Kler, P. A., and Cosimo, A. (2011). "Parallel Distributed Computing Using Python." *Advances in Water Resources*, 34(9), 1124–1139.

[43] Demmel, J. W. (1997). *Applied Numerical Linear Algebra.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

[44] Dombre, T., Frisch, U., Greene, J. M., Hénon, M., Mehr, A., and Soward, A. M. (1986). "Chaotic Streamlines in The ABC Flows." *Journal of Fluid Mechanics*, 167, 353–391.

[45] Dongarra, J. J., Croz, J. D., Hammarling, S., and Duff, I. S. (1990). "A Set of Level 3 Basic Linear Algebra Subprograms." *ACM Trans. Math. Softw.*, 16(1), 1–17.

[46] Duvaut, G. and Lions, J. L. (1976). *Inequalities in Mechanics and Physics.* Springer-Verlag, Berlin, Germany.

[47] Elman, H. C., Silvester, D. J., and Wathen, A. J. (2006). "Finite Elements and Fast Iterative Solvers." *Journal of Fluid Mechanics*, 557(1), 474–475.

[48] EPA, U. (2004). "Cleaning Up the Nation's Waste Sites: Markets and Technology Trends." *Report No. EPA 542-R-04-015.*

[49] Evans, L. C. (1998). *Partial Differential Equations.* American Mathematical Society, Providence, Rhode Island, USA.

[50] Facchinei, F. and Pang, J.-S. (2003). *Finite-Dimensional Variational Inequalities and Complementarity Problems. Volume I.* Springer-Verlag, New York, USA.

[51] Falgout, R. (2006). “HYPRE users manual.” *Report No. Revision 2.0.0*, Lawrence Livermore National Laboratory.

[52] Fichera, G. (1964). *Problemi Elastostatici con Vincoli Unilaterali: Il Problema di Signorini con Ambigue Condizioni al Contorno*. Accademia nazionale dei Lincei.

[53] Fichera, G. (1965). *Linear Elliptic Differential Systems and Eigenvalue Problems*. Number 8. Springer.

[54] Fischer, A. (1992). “A Special Newton-Type Optimization Method.” *Optimization*, 24, 269–284.

[55] Gaston, D., Newman, C., Hansen, G., and Lebrun-Grandie, D. (2009). “MOOSE: A Parallel Computational Framework for Coupled Systems of Nonlinear Equations.” *Nuclear Engineering and Design*, 239(10), 1768–1778.

[56] Genty, A. and Potier, C. L. (2011). “Maximum and Minimum Principles for Radionuclide Transport Calculations in Geological Radioactive Waste Repository: Comparison Between a Mixed Hybrid Finite Element Method and Finite Volume Element Discretizations.” *Transport in Porous Media*, 88, 65–86.

[57] Geuzaine, C. and Remacle, J. F. (2009). “GMSH: A Three-Dimensional Finite Element Mesh Generator with Built-In Pre- and Post-Processing Facilities.” *International Journal for Numerical Methods in Engineering*, 79, 1309–1331.

[58] Gibbs, J. W. (1898). “Fourier’s Series.” *Nature*, 59(1522), 200.

[59] Gibbs, J. W. (1899). “Fourier’s Series.” *Nature*, 59(1539), 606.

[60] Gilbarg, D. and Trudinger, N. S. (2001). *Elliptic Partial Differential Equations of Second Order*. Springer, New York, USA.

[61] Glowinski, R. (1984). *Numerical Methods for Nonlinear Variational Problems*. Springer-Verlag, Berlin, Germany.

[62] Graser, C. and Kornhuber, R. (2009). “Multigrid Methods for Obstacle Problems.” *Journal of Computational Mathematics*, 27, 1–44.

[63] Grcar, J. F. (2011). “Mathematicians of Gaussian Elimination.” *Notices of the American Mathematical Society*, 58(6), 782–792.

[64] Gropp, W., Lusk, E., and Skjellum, A. (1999). *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, Vol. 1.

[65] Gropp, W. D., Kaushik, D. K., Keyes, D. E., and Smith, B. F. (1999). “Toward Realistic Performance Bounds for Implicit CFD Codes.” *Proceedings of Parallel CFD ‘99*. Elsevier, 233–240.

[66] Grossmann, C., Roos, H. G., and Stynes, M. (2007). *Numerical Treatment of Partial Differential Equations*. Springer, Berlin, Germany.

[67] Hammond, G. E. and Lichtner, P. C. (2010). “Field-Scale Model for the Natural Attenuation of Uranium at the Hanford 300 Area using High-Performance Computing.” *Water Resources Research*, 46, W09602.

[68] Han, W. and Reddy, B. D. (2012). *Plasticity: Mathematical Theory and Numerical Analysis*, Vol. 9. Springer Science & Business Media.

[69] Harari, I. (2004). “Stability of Semidiscrete Formulations for Parabolic Problems at Small Time Steps.” *Computer Methods in Applied Mechanics and Engineering*, 193(15), 1491–1516.

[70] Harp, D. R. and Vesselinov, V. V. (2013). “Contaminant Remediation Decision Analysis Using Information Gap Theory.” *Stochastic Environmental Research and Risk Assessment*, 27, 159–168.

[71] Heikoop, J. M., Johnson, T. M., Birdsell, K. H., Longmire, P., Hickmott, D. D., Jacobs, E. P., Broxton, D. E., Katzman, D., Vesselinov, V. V., Ding, M., Vanimana,

D. T., Reneaua, S. L., Goering, T. J., Glessnerb, J., and Basu, A. (2014). “Isotopic Evidence for Reduction of Anthropogenic Hexavalent Chromium in Los Alamos National Laboratory Groundwater.” *Chemical Geology*, 373, 1–9.

[72] Hendrickson, B. and Leland, R. (1995a). “A Multilevel Algorithm for Partitioning Graphs.” *Supercomputing ’95: Proceedings of the 1995 ACM/IEEE Conference on Supercomputing (CDROM)*, New York. ACM Press, 28.

[73] Hendrickson, B. and Leland, R. W. (1995b). “A Multi-Level Algorithm For Partitioning Graphs..” *SC*, 95(28).

[74] Heroux, M. A., Bartlett, R. A., Howle, V. E., Hoekstra, R. J., Hu, J. J., Kolda, T. G., Lehoucq, R. B., Long, K. R., Pawlowski, R. P., Phipps, E. T., Salinger, A. G., Thornquist, H. K., Tuminaro, R. S., Willenbring, J. M., Williams, A., and Stanley, K. S. (2005). “An Overview of the Trilinos project.” *ACM Trans. Math. Softw.*, 31(3), 397–423.

[75] Hjelmstad, K. D. (2005). *Fundamentals of Structural Mechanics*. Springer Science+Business Media, Inc., New York, USA, second edition.

[76] Hlavacek, I., Haslinger, J., Necas, J., and Lovisek, J. (2012). *Solution of Variational Inequalities in Mechanics*, Vol. 66. Springer Science & Business Media.

[77] Homolya, M. and Ham, D. A. (2016). “A Parallel Edge Orientation Algorithm for Quadrilateral Meshes.” *SIAM Journal on Scientific Computing*, 38, S48–S61.

[78] Huang, W. and Wang, Y. (2015). “Discrete Maximum Principle for the Weak Galerkin Method for Anisotropic Diffusion Problems.” *Communications in Computational Physics*, 18(1), 65–90.

[79] Karra, S., Painter, S. L., and Lichtner, P. C. (2014). “Three-Phase Numerical

Model for Subsurface Hydrology in Permafrost-Affected Regions (PFLOTRAN-ICE v1.0)." *The Cryosphere*, 8(5), 1935–1950.

[80] Karypis, G. and Kumar, V. (1999). "A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs." *SIAM Journal on Scientific Computing*, 20, 359–392.

[81] Kelkar, S., Lewis, K., Karra, S., Zyvoloski, G., Rapaka, S., Viswanathan, H., Mishra, P. K., Chu, S., Coblentz, D., and Pawar, R. (2014). "A Simulator for Modeling Coupled Thermo-Hydro-Mechanical Processes in Subsurface Geological Media." *International Journal of Rock Mechanics and Mining Sciences*, 70, 569–580.

[82] Keyes, D. E., McInnes, L. C., Woodward, C., Gropp, W., Myra, E., Pernice, M., Bell, J., Brown, J., Clo, A., Connors, J., Constantinescu, E., Estep, D., Evans, K., Farhat, C., Hakim, A., Hammond, G., Hansen, G., Hill, J., Isaac, T., Jiao, X., Jordan, K., Kaushik, D., Kaxiras, E., Koniges, A., Lee, K., Lott, A., Lu, Q., Magerlein, J., Maxwell, R., McCourt, M., Mehl, M., Pawlowski, R., Randles, A. P., Reynolds, D., Rivière, B., ode, U. R., Scheibe, T., Shadid, J., Sheehan, B., Shephard, M., Siegel, A., Smith, B., Tang, X., Wilson, C., and Wohlmuth, B. (2013). "Multiphysics Simulations." *The International Journal of High Performance Computing Applications*, 27(1), 4–83.

[83] Kikuchi, N. and Oden, J. T. (1988). *Contact Problems in Elasticity: A Study of Variational Inequalities and Finite Element Methods*, Vol. 8. SIAM.

[84] Kinderlehrer, D. and Stampacchia, G. (2000). *An Introduction to Variational Inequalities and Their Applications*. SIAM Classics in Applied Mathematics, New York, USA.

[85] Kirby, R. C. (2012). *FIAT: Numerical Construction of Finite Element Basis Functions*, chapter 13. Springer.

[86] Kirk, B. S., Peterson, J. W., Stogner, R. H., and Carey, G. F. (2006). "`LibMesh`: A C++ Library for Parallel Adaptive Mesh Refinement/Coarsening Simulations." *Engineering with Computers*, 22(3–4), 237–254. `http://dx.doi.org/10.1007/s00366-006-0049-3`.

[87] Knepley, M. G. and Bardhan, J. P. (2015). "Work/Precision Tradeoffs in Continuum Models of Biomolecular Electrostatics." *Proceedings of ASME 2015 International Mechanical Engineering Congress & Exposition*, Vol. 9. V009T12A04.

[88] Knepley, M. G., Brown, J., Rupp, K., and Smith, B. F. (2013). "Achieving High Performance with Unified Residual Evaluation." *arXiv:1309.1204*.

[89] Knepley, M. G. and Karpeev, D. A. (2009). "Mesh Algorithms for PDE with Sieve I: Mesh Distribution." *Scientific Programming*, 17, 215–230.

[90] Knepley, M. G., Rupp, K., and Terrel, A. R. (2016). "Finite Element Integration with Quadrature on the GPU.

[91] Knepley, M. G. and Terrel, A. R. (2013). "Finite Element Integration on GPUs." *ACM Transactions on Mathematical Software*, 39(2).

[92] Lange, M., Knepley, M. G., and Gorman, G. J. (2015). "Flexible, Scalable Mesh and Data Management Using PETSc DMPlex." *Proceedings of the 3rd International Conference on Exascale Applications and Software*, EASC '15. University of Edinburgh, 71–76.

[93] Lawrence Livermore National Laboratory. *HYPRE: High Performance Preconditioners*. `http://www.llnl.gov/CASC/hypre/`.

[94] Lester, D. R., Metcalfe, G., and Trefry, M. G. (2013). “Is chaotic advection inherent to porous media flow?.” *Physical Review Letters*, 111(17), 174101.

[95] Li, J. and Riviére, B. (2015a). “High Order Discontinuous Galerkin Method for Simulating Miscible Flooding in Porous Media.” *Computational Geoscience*, 19, 1251–1268.

[96] Li, J. and Riviére, B. (2015b). “Numerical Solutions of the Incompressible Miscible Displacement Equations in Heterogeneous Media.” *Computer Methods in Applied Mechanics and Engineering*, 292, 107–121.

[97] Li, J. and Riviére, B. (2016). “Numerical Modeling of Miscible Viscous Fingering Instabilities by High-Order Methods.” *Transport in Porous Media*, 113, 607âĂŞ628.

[98] Li, X. S. and Demmel, J. W. (2003). “SuperLU_DIST: A Scalable Distributed-Memory Sparse Direct Solver for Unsymmetric Linear Systems.” *ACM Transactions on Mathematical Software (TOMS)*, 29(2), 110–140.

[99] Lichtner, P. C., Hammond, G. E., Lu, C., Karra, S., Bisht, G., Andre, B., Mills, R. T., and Kumar, J. (2013). “PFLOTRAN Web Page. http://www.pflotran.org.

[100] Lichtner, P. C., Hammond, G. E., Lu, C., Karra, S., Bisht, G., Andre, B., Mills, R. T., and Kumar, J. (2015). “PFLOTRAN User Manual: A Massively Parallel Reactive Flow and Transport Model for Describing Surface and Subsurface Processes.” *Report No. Report No.: LA-UR-15-20403*, Los Alamos National Laboratory.

[101] Lichtner, P. C. and Karra, S. (2014). “Modeling Multiscale-Multiphase-Multicomponent Reactive Flows in Porous Media: Application to CO_2 Sequestration and Enhanced Geothermal Energy using PFLOTRAN.” *Computational Models for CO_2 Geo-sequestration & Compressed Air*

Energy Storage, R. Al-Khoury and J. Bundschuh, eds., CRC Press, http://www.crcnetbase.com/doi/pdfplus/10.1201/b16790-6, 81–136.

[102] Lin, C. J. and Moré, J. (1999). "Newton's Method for Large Bound-Constrained Optimization Problems." *SIAM Journal on Optimization*, 9, 1100–1127.

[103] Lions, J. L. and Stampacchia, G. (1967). "Variational Inequalities." *Communications on Pure and Applied Mathematics*, 20(3), 493–519.

[104] Lipnikov, K., Shashkov, M., Svyatskiy, D., and Yassilevski, Y. (2007). "Monotone Finite Volume Schemes for Diffusion Equations on Unstructured Triangular and Shape-Regular Polygonal Meshes." *Journal of Computational Physics*, 227, 492–512.

[105] Liska, R. and Shashkov, M. (2008). "Enforcing the Discrete Maximum Principle for Linear Finite Element Solutions for Elliptic Problems." *Communications in Computational Physics*, 3, 852–877.

[106] Lo, Y. J., Williams, S., Straalen, B. V., Ligocki., T. J., Cordery, M. J., Wright, N. J., Hall, M. W., and Oliker, L. (2015). "Roofline: An Insightful Visual Performance Model for Multicore Architectures." *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, 8966, 129–148.

[107] Logg, A. (2009). "Efficient Representation of Computational Meshes." *International Journal of Computational Science and Engineering*, 4, 283–295.

[108] Logg, A., Mardal, K. A., and Wells, G. N. (2012). *Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book*, Vol. 84. Springer Science & Business Media.

[109] Luca, T. D., Facchinei, F., and Kanzow, C. (1996). "A Semismooth Equation

Approach to the Solution of Nonlinear Complementarity Problems." *Mathematical Programming*, 75, 407–439.

[110] Luporini, F., Ham, D. A., and Kelly, P. H. J. (2016). "An Algorithm for the Optimization of Finite Element Integration Loops." *Submitted to ACM TOMS*.

[111] Luporini, F., Varbanescu, A. L., Rathgeber, F., Bercea, G. T., Ramanujam, J., Ham, D. A., and Kelly, P. H. J. (2015). "Cross-Loop Optimization of Arithmetic Intensity for Finite Element Local Assembly." *ACM Transactions on Architecture and Code Optimization*, 11(4), 57:1–57:25.

[112] Markall, G. R., Rathgeber, F., Mitchell, L., Loriant, N., Bertolli, C., Ham, D. A., and Kelly, P. H. J. (2013). "Performance-Portable Finite Element Assembly Using PyOP2 and FEniCS." *28th International Supercomputing Conference, ISC, Proceedings*, J. M. Kunkel, T. Ludwig, and H. W. Meuer, eds., Vol. 7905 of *Lecture Notes in Computer Science*. Springer, 279–289.

[113] May, D. A., Brown, J., and Laetitia, L. L. (2014). "pTatin3D: High-Performance Methods for Long-Term Lithospheric Dynamics." *Proceedings of the International Conference for High Performance Computing, Network, Storage and Analysis*, SC '14. IEEE Press, 274–284.

[114] McCalpin, J. (1995). "STREAM: Sustainable Memory Bandwidth in High Performance Computers. https://www.cs.virginia.edu/stream/.

[115] McKee, S. A. (2004). "Reflections on the Memory Wall." *Proceedings of the 1st conference on Computing frontiers*, ACM. 162.

[116] McRae, A. T. T., Bercea, G. T., Mitchell, L., Ham, D. A., and Cotter, C. J. (2016). "Automated Generation and Symbolic Manipulation of Tensor Product Finite Elements." *SIAM Journal on Scientific Computing*, 38, S25–S47.

[117] Mifflin, R. (1977). "Semismooth and Semiconvex Functions in Constrained Optimization." *SIAM Journal on Control and Optimization*, 15, 957–972.

[118] Miller, C. T., Dawson, C. N., Farthing, M. W., Hou, T. Y., Huang, J., Kees, C. E., Kelley, C. T., and Langtangen", H. P. (2013). "Numerical simulation of water resources problems: Models, methods, and trends." *Advances in Water Resources*, 51, 405–437.

[119] Mucci, P. J., Browne, S., Deane, C., and Ho, G. (1999). "PAPI: A Portable Interface to Hardware Performance Counters." *Proceedings of the department of defense HPCMP users group conference*, Vol. 710.

[120] Mudunuru, M. K. and Nakshatrala, K. B. (2016). "On Enforcing Maximum Principles and Achieving Element-Wise Species Balance for Advection-Diffusion-Reaction Equations under the Finite Element Method." *Journal of Computational Physics*, 305, 448–493.

[121] Mudunuru, M. K. and Nakshatrala, K. B. (2017). "On Mesh Restrictions to Satisfy Comparison Principles, Maximum Principles, and the Non-Negative Constraint: Recent Developments and New Results." *Mechanics of Advanced Materials and Structures*, 24, 556–590.

[122] Mudunuru, M. K., Shabouei, M., and Nakshatrala, K. B. (2015). "On Local and Global Species Conservation Errors for Nonlinear Ecological Models and Chemical Reacting Flows." *ASME 2015 International Mechanical Engineering Congress and Exposition*, American Society of Mechanical Engineers. V009T12A018–V009T12A018.

[123] Munson, T., Sarich, J., Wild, S., Benson, S., and McInnes, L. C. (2014). "Toolkit for Advanced Optimization (TAO) Users Manual." *Report No. ANL/MCS-TM-322 - Revision 3.5*, Argonne National Laboratory.

[124] Munson, T. S., Facchinei, F., Ferris, M. C., Fischer, A., and Kanzow, C. (2001). "The Semismooth Algorithm for Large Scale Complementarity Problems." *INFORMS Journal on Computing*, 13, 294–311.

[125] Murphy, M. F., Golub, G. H., and Wathen, A. J. (2000). "A Note on Preconditioning for Indefinite Linear Systems." *SIAM Journal on Scientific Computing*, 21(6), 1969–1972.

[126] Murphy, R. (2007). "On the Effects of Memory Latency and Bandwidth on Supercomputer Application Performance." *Workload Characterization, 2007. IISWC 2007. IEEE 10th International Symposium on*, IEEE. 35–43.

[127] Nagarajan, H. and Nakshatrala, K. B. (2011). "Enforcing the Non-Negativity Constraint and Maximum Principles for Diffusion with Decay on General Computational Grids." *International Journal for Numerical Methods in Fluids*, 67, 820–847.

[128] Nagurney, A. (2002). "Variational Inequalities." *Report no.*, Isenberg School of Management, University of Massachusetts Amherst.

[129] Nagurney, A. and Zhang, D. (2012). *Projected Dynamical Systems and Variational Inequalities with Applications*, Vol. 2. Springer Science & Business Media.

[130] Nakshatrala, K. B., Mudunuru, M. K., and Valocchi, A. J. (2013). "A Numerical Framework for Diffusion-Controlled Bimolecular-Reactive Systems to Enforce Maximum Principles and the Non-Negative Constraint." *Journal of Computational Physics*, 253, 278–307.

[131] Nakshatrala, K. B., Nagarajan, H., and Shabouei, M. (2016). "A Numerical Methodology for Enforcing Maximum Principles and the Non-Negative Constraint for Transient Diffusion Equations." *Communications in Computational Physics*, 19, 53–93.

[132] Nakshatrala, K. B. and Valocchi, A. J. (2009). "Non-negative Mixed Finite Element Formulations for a Tensorial Diffusion Equation." *Journal of Computational Physics*, 228, 6726–6752.

[133] Pal, R. K., Abedi, R., Madhukar, A., and Haber, R. B. (2016). "Adaptive Spacetime Discontinuous Galerkin Method for Hyperbolic Advection-Diffusion with a Non-Negativity Constraint." *International Journal for Numerical Methods in Engineering*, 105, 963–989.

[134] Payette, G. S., Nakshatrala, K. B., and Reddy, J. N. (2012). "On the Performance of High-Order Finite Elements with Respect to Maximum Principles and the Non-Negative Constraint for Diffusion-Type Equations." *International Journal for Numerical Methods in Engineering*, 91, 742–771.

[135] Potier, C. L. (2005). "Finite Volume Monotone Scheme for Highly Anisotropic Diffusion Operators on Unstructured Triangular Meshes." *Comptes Rendus Mathematique*, 341, 787–792.

[136] Pruess, K. (2004). "The TOUGH Codes—A Family of Simulation Tools for Multiphase Flow and Transport Processes in Permeable Media." *Vadose Zone Journal*, 3(3), 738–746.

[137] Pyzara, A., Bylina, B., and Bylina, J. (2011). "The Influence of a Matrix Condition Number on Iterative Methods' Convergence." *Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on*, IEEE. 459–464.

[138] Qi, L. (1993). "Convergence Analysis of Some Algorithms for Solving Nonsmooth Equations." *Mathematics of Operations Research*, 18, 227–244.

[139] Qi, L. and Sun, J. (1993). "A Nonsmooth Version of Newton's Method." *Mathematical Programming*, 58, 353–368.

[140] Rajovic, N., Rico, A., Puzovic, N., Adeniyi-Jones, C., and Ramirez, A. (2014). "Tibidabo1: Making the Case for an ARM-based {HPC} system." *Future Generation Computer Systems*, 36, 322 – 334.

[141] Rathgeber, F., Ham, D. A., Mitchell, L., Lange, M., Luporini, F., McRae, A. T. T., Bercea, G. T., Markall, G. R., and Kelly, P. H. J. (2016). "Firedrake: Automating the Finite Element Method by Composing Abstractions." *ACM TOMS*, 43, 24:1–24:27.

[142] Rathgeber, F., Markall, G. R., Mitchell, L., Loriant, N., Ham, D. A., Bertolli, C., and Kelly, P. H. J. (2012). "PyOP2: A High-Level Framework for Performance-Portable Simulations on Unstructured Meshes." *High Performance Computing, Networking Storage and Analysis, SC Companion:*, Los Alamitos, CA, USA. IEEE Computer Society, 1116–1123.

[143] Raviart, P. A. and Thomas, J. M. (1977). "A Mixed Finite Element Method for 2-nd Order Elliptic Problems." *Mathematical aspects of finite element methods*, Springer, 292–315.

[144] Riviére, B. and Wheeler, M. (2002). "Discontinuous Galerkin Methods for and Transport in Porous Media." *Communications in Numerical Methods in Engineering*, 18, 63–68.

[145] Rodrigues, J. (1987). *Obstacle Problems in Mathematical Physics*, Vol. 134. Elsevier.

[146] Rognes, M. E., Kirby, R. C., and Logg, A. (2009). "Efficient Assembly of H(div) and H(curl) Conforming Finite Elements." *SIAM Journal on Scientific Computing*, 31(6), 4130–4151.

[147] Sala, M., Hu, J. J., and Tuminaro, R. S. (2004). "ML3.1 Smoothed Aggregation User's Guide." *Report No. SAND2004-4821*, Sandia National Laboratories.

[148] Schoof, C. (2006). "A Variational Approach to Ice Stream Flow." *Journal of Fluid Mechanics*, 556, 227–251.

[149] Schoof, C. and Hindmarsh, R. (2010). "Thin-Film Flows with Wall Slip: An Asymptotic Analysis of Higher Order Glacier Flow Models." *Quarterly Journal of Mechanics and Applied Mathematics*, 63, 73–114.

[150] Schulz, M., Galarowicz, J., Maghrak, D., Hachfeld, W., Montoya, D., and Cranford, S. (2008). "Open| SpeedShop: An Open Source Infrastructure for Parallel Performance Analysis." *Scientific Programming*, 16(2-3), 105–121.

[151] Shahbazi, K. (2005). "Short Note: An Explicit Expression for the Penalty Parameter of the Interior Penalty Method." *Journal of Computational Physics*, 205(2), 401–407.

[152] Sheng, Z. and Yuan, G. (2016). "A new nonlinear finite volume scheme preserving positivity for diffusion equations." *Journal of Computational Physics*, 315, 182–193.

[153] Signorini, A. (1933). "Sopra Alcune Questioni di Statica Dei Sistemi Continui." *Annali della Scuola Normale Superiore di Pisa-Classe di Scienze*, 2(2), 231–251.

[154] Signorini, A. (1959). "Questioni di Elasticità Non Linearizzata e Semilinearizzata.." *Rendiconti Di Matematica e Delle Sue Applicazioni, V. Serie*, 18, 95–139.

[155] Sinharoy, B., Norstrand, J. A. V., Eickemeyer, R. J., Le, H. Q., Leenstra, J., Nguyen, D. Q., Konigsburg, B., Ward, K., Brown, M. D., Moreira, J. E., Levitan, D., Tung, S., Hrusecky, D., Bishop, J. W., Gschwind, M., Boersma, M., Kroener, M., Kaltenbach, M., Karkhanis, T., and Fernsler, K. M. (2015). "IBM POWER8 Processor Core Microarchitecture." *IBM Journal of Research and Development*, 59(1), 2:1–2:21.

[156] Smith, B. F., Bjørstad, P., and Gropp, W. D. (1996). *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press.

[157] Stalkup, F. I. (1983). "Miscible Displacement." *Society of Petroleum Engineers*.

[158] The HDF Group (1997-2015). "Hierarchical Data Format, Version 5. http://www.hdfgroup.org/HDF5/.

[159] Turner, D. Z., Nakshatrala, K. B., Martinez, M. J., and Notz, P. K. (2011). "Modeling Subsurface Water Resource Systems Involving Heterogeneous Porous Media Using the Variational Multiscale Formulation." *Journal of Hydrology*, 428-429, 1–14.

[160] Ulbrich, M. (2011). *Semismooth Newton Methods for Variational Inequalities and Constrained Optimization Problems in Function Spaces*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

[161] von der Schulenburg, D. A., Pintelon, T. R. R., Picioreanu, C., Loosdrecht, M. C. V., and Johns, M. L. (2009). "Three-Dimensional Simulations of Biofilm Growth in Porous Media." *AIChE Journal*, 55(2), 494–504.

[162] Weaver, V. M., Terpstra, D., and Moore, S. (2013). "Non-Determinism and Overcount on Modern Hardware Performance Counter Implementations." *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 215–224.

[163] Werth, C. E. K. C. J. and Valocchi, A. J. (2005). "Pore-Scale Simulation of Biomass Growth Along the Transverse Mixing Zone of a Model Two-Dimensional Porous Medium." *Water Resources Research*, 41(7).

[164] Williams, S., Waterman, A., and Patterson, D. (2009). “Roofline: An Insightful Visual Performance Model for Multicore Architectures.” *Communications of the ACM*, 54, 65–76.

[165] Wulf, W. A. and McKee, S. A. (1995). “Hitting the Memory Wall: Implications of the Obvious.” *ACM SIGARCH computer architecture news*, 23(1), 20–24.

[166] Zhao, X. H., Kwek, K. H., Li, J. B., and Huang, K. L. (1993). “Chaotic and Resonant Streamlines in the ABC Flow.” *SIAM Journal on Applied Mathematics*, 53, 71–77.

[167] Zyvoloski, G. (2007). “FEHM: A Control Volume Finite Element Code for Simulating Subsurface Multi-Phase Multi-Fluid Heat and Mass Transfer.” *Los Alamos Unclassified Report LA-UR-07-3359.*

Appendix A. Firedrake Project

The Firedrake project [141, 110, 111] is a python-based library that provides an automated system for the solution of partial differential equations using the finite element method. Like the FEniCS Project [108, 4], it is also built upon several scientific packages and can employ parallel computing tools across either CPUs or GPUs to obtain the solution. Two of its main leveraged components are the Unified Form Language (UFL) [5], used to declare finite element discretizations of variational forms, and the PyOP2 system [142, 112], used for the parallel assembly of the finite element discrete formulations. The main difference between the FEniCS and Firedrake project is that all data structures, linear solvers, non-linear solvers, and optimization solvers for the latter are provided entirely by the PETSc and TAO libraries. The mesh can either be generated internally or imported from third party mesh generators like GMSH [57], and the parallel partitioning of the mesh is achieved through packages like Chaco [72]. Another important feature utilized in this dissertation is extruded meshes. The internal mesh algorithm generates and partitions a 2D quadrilateral base mesh and is extruded into a hexahedron mesh using the algorithms listed in [77, 116]. To facilitate the readers to be able to reproduce the results presented in this dissertation, we provided some useful Firedrake-related files below.

Listing A.1: 2D GAL diffusion example

```python
# Load firedrake environment
from firedrake import *

# Create mesh
mesh = UnitSquareMesh(200,200,quadrilateral=True)

# Function spaces
P = FunctionSpace(mesh, 'Lagrange', 1)
Q = TensorFunctionSpace(mesh, 'Lagrange', 1)
u = TrialFunction(P)
v = TestFunction(P)

```

```
# Bounds
cmin = 0.0
cmax = PETSc.INFINITY
lb = Function(P)
lb.assign(cmin)
ub = Function(P)
ub.assign(cmax)

# Diffusion tensor
eps = 1e-4
D = interpolate(Expression(('eps*x[0]*x[0]+x[1]*x[1]','-(1-eps)*x[0]*x[1]',
     '-(1-eps)*x[0]*x[1]','eps*x[1]*x[1]+x[0]*x[0]'),eps=eps),Q)

# Forcing function
lb = 0.375
ub = 0.625
f = interpolate(Expression(('x[0] ≥ lb && x[0]≤ ub && x[1] ≥ lb &&
     x[1] ≤ ub ? 1.0 : 0.0'),lb=lb,ub=ub),P)

# GAL formulation
a = dot(D * grad(u), grad(v)) * dx
L = v * f * dx

# Homogeneous boundary conditions
bcs = DirichletBC(P, Constant(0.0), (1,2,3,4))

# Assemble coefficient matrix
A = assemble(a, bcs=bcs)

# Assemble forcing vector
tmp = Function(P)
bcs.apply(tmp)
b = Function(P)
bfree = assemble(L)
rhs_bcs = assemble(action(a,tmp))
b.assign(bfree - rhs_bcs)
bcs.apply(b)

# Create PETSc solver
initial_solver = PETSc.KSP().create(PETSc.COMM_WORLD)
initial_solver.setOptionsPrefix("initial_")
initial_solver.setOperators(A.M.handle)
initial_solver.setFromOptions()
initial_solver.setUp()

# Solve problem
solution = Function(P)
with b.dat.vec_ro as b_vec, solution.dat.vec as sol_vec:
  initial_solver.solve(b_vec,sol_vec)
```

Listing A.2: 2D SUPG advection-diffusion example

```
# Load firedrake environment
from firedrake import *

# Load GMSH file
mesh = Mesh('square_hole.msh')

# Function spaces
P = FunctionSpace(mesh, 'Lagrange', 1)
V = VectorFunctionSpace(mesh, 'Lagrange', 1)
u = TrialFunction(P)
v = TestFunction(P)

# Bounds
cmin = 0.0
cmax = 1.0
lb = Function(P)
lb.assign(cmin)
ub = Function(P)
ub.assign(cmax)

# Velocity field
velocity = interpolate(Expression(('cos(2*pi*x[1]*x[1])',
    'sin(2*pi*x[0])+cos(2*pi*x[0]*x[0])')),V)

# Diffusion tensor
alphaT = Constant(1e-5)
alphaL = Constant(1e-1)
alphaD = Constant(1e-9)
normv = sqrt(dot(velocity,velocity))
Id = Identity(mesh.geometric_dimension())
D = (alphaD + alphaT*normv)*Id +
    (alphaL - alphaT)*outer(velocity,velocity)/normv

# Forcing function
f = Constant(0.0)

# SUPG weak form
h = CellSize(mesh)
Pe = h/(2*normv)*dot(velocity,grad(v))
ar = Pe*(dot(velocity,grad(u)) - div(D*grad(u)))*dx
a = ar + v*dot(velocity,grad(u))*dx + dot(grad(v),D*grad(u))*dx
a = dot(D*grad(u), grad(v))*dx
L = (v + Pe)*f*dx

# Boundary conditions
bc1 = DirichletBC(P, Constant(0.0), (12,13,14,15)) # Outer square
bc2 = DirichletBC(P, Constant(1.0), (16,17,18,19)) # Inner square
bcs = [bc1,bcs2]

# Homogeneous boundary conditions
bcs = DirichletBC(P, Constant(0.0), (1,2,3,4))

# Assemble coefficient matrix
A = assemble(a, bcs=bcs)

# Assemble forcing vector
```

```
tmp = Function(P)
for bc in bcs:
  bc.apply(tmp)
b = Function(P)
bfree = assemble(L)
rhs_bcs = assemble(action(a,tmp))
b.assign(bfree - rhs_bcs)
for bc in bcs:
  bc.apply(b)

# Create PETSc solver
initial_solver = PETSc.KSP().create(PETSc.COMM_WORLD)
initial_solver.setOptionsPrefix("initial_")
initial_solver.setOperators(A.M.handle)
initial_solver.setFromOptions()
initial_solver.setUp()

# Solve problem
solution = Function(P)
with b.dat.vec_ro as b_vec, solution.dat.vec as sol_vec:
  initial_solver.solve(b_vec,sol_vec)
```

Listing A.3: 3D DG advection-diffusion example

```
# Load firedrake environment
from firedrake import *

# Number of elements in each spatial dimension
seed = 40

# 2D base mesh
mesh = UnitSquareMesh(seed,seed,quadrilateral=True)
# Extruded mesh
mesh = ExtrudedMesh(meshbase,seed)

# Function spaces
P = FunctionSpace(mesh, 'DG', 1)
V = VectorFunctionSpace(mesh, 'DG', 1)
u = TrialFunction(P)
v = TestFunction(P)

# Bounds
cmin = 0.0
cmax = PETSc.INFINITY
lb = Function(P)
lb.assign(cmin)
ub = Function(P)
ub.assign(cmax)

# Velocity field
velocity = interpolate(Expression(('0.3*sin(2*pi*x[2])+cos(3*pi*x[1])',
    '0.65*sin(2*pi*x[0])+0.3*cos(5*pi*x[2])',
    'sin(4*pi*x[1])+0.65*cos(6*pi*x[0])')),V)

# Diffusion tensor
alphaT = Constant(1e-5)
alphaL = Constant(1e-1)
alphaD = Constant(1e-9)
normv = sqrt(dot(velocity,velocity))
Id = Identity(mesh.geometric_dimension())
D = (alphaD + alphaT*normv)*Id +
    (alphaL - alphaT)*outer(velocity,velocity)/normv

# Forcing function
f1 = interpolate(Expression(('x[0] ≥ 0.4 && x[0] ≤ 0.5 && x[1] ≥ 0.2 &&
    x[1] ≤ 0.3 && x[2] ≥ 0.1 && x[2] ≤ 0.2 ? 1.0 : 0.0')),P)
f2 = interpolate(Expression(('x[0] ≥ 0.8 && x[0] ≤ 0.9 && x[1] ≥ 0.4 &&
    x[1] ≤ 0.5 && x[2] ≥ 0.2 && x[2] ≤ 0.3 ? 1.0 : 0.0')),P)
f3 = interpolate(Expression(('x[0] ≥ 0.5 && x[0] ≤ 0.6 && x[1] ≥ 0.7 &&
    x[1] ≤ 0.8 && x[2] ≥ 0.3 && x[2] ≤ 0.4 ? 1.0 : 0.0')),P)
f4 = interpolate(Expression(('x[0] ≥ 0.3 && x[0] ≤ 0.4 && x[1] ≥ 0.5 &&
    x[1] ≤ 0.6 && x[2] ≥ 0.2 && x[2] ≤ 0.3 ? 1.0 : 0.0')),P)
f5 = interpolate(Expression(('x[0] ≥ 0.5 && x[0] ≤ 0.6 && x[1] ≥ 0.2 &&
    x[1] ≤ 0.3 && x[2] ≥ 0.6 && x[2] ≤ 0.7 ? 1.0 : 0.0')),P)
f6 = interpolate(Expression(('x[0] ≥ 0.6 && x[0] ≤ 0.7 && x[1] ≥ 0.5 &&
    x[1] ≤ 0.6 && x[2] ≥ 0.7 && x[2] ≤ 0.8 ? 1.0 : 0.0')),P)
f7 = interpolate(Expression(('x[0] ≥ 0.4 && x[0] ≤ 0.5 && x[1] ≥ 0.7 &&
    x[1] ≤ 0.8 && x[2] ≥ 0.8 && x[2] ≤ 0.9 ? 1.0 : 0.0')),P)
f8 = interpolate(Expression(('x[0] ≥ 0.1 && x[0] ≤ 0.2 && x[1] ≥ 0.4 &&
    x[1] ≤ 0.5 && x[2] ≥ 0.7 && x[2] ≤ 0.8 ? 1.0 : 0.0')),P)
f = f1 + f2 + f3 + f4 + f5 + f6 + f7 + f8

# Parameters
h = Constant(1/float(seed)) # h-size
gamma = Constant(8/3) # Penalty term
n = FacetNormal(mesh) # Unit outward normal
vn = 0.5*(dot(velocity,n) + abs(dot(velocity,n))) # Upwinding term

# DG weak formulation
a = inner(D * grad(u), grad(v)) * dx(degree=(3,3)) \
    - dot(jump(v,n),avg(D*grad(u)))*(dS_h + dS_v) \
    - dot(avg(D*grad(v)),jump(u,n))*(dS_h + dS_v) \
    + gamma/h*dot(jump(v,n),jump(u,n))*(dS_h + dS_v) \
    - dot(grad(v),velocity*u)*dx(degree=(3,3)) \
    + dot(jump(v),vn('+')*u('+')-vn('-')*u('-'))*(dS_h+dS_v) \
    + dot(v, vn*u)*(ds_v+ds_t+ds_b)
L = v * f * dx(degree=(3,3))

# Boundary conditions
bc1 = DirichletBC(V, Constant(0.0), (1,2,3,4), method="geometric")
bc2 = DirichletBC(V, Constant(0.0), "bottom", method="geometric")
bc3 = DirichletBC(V, Constant(0.0), "top", method="geometric")
bcs = [bc1, bc2, bc3]

# Homogeneous boundary conditions
bcs = DirichletBC(P, Constant(0.0), (1,2,3,4))

# Assemble coefficient matrix
```

```
A = assemble(a, bcs=bcs)

# Assemble forcing vector
tmp = Function(P)
for bc in bcs:
  bc.apply(tmp)
b = Function(P)
bfree = assemble(L)
rhs_bcs = assemble(action(a,tmp))
b.assign(bfree - rhs_bcs)
for bc in bcs:
  bc.apply(b)

# Create PETSc solver
initial_solver = PETSc.KSP().create(PETSc.COMM_WORLD)
initial_solver.setOptionsPrefix("initial_")
initial_solver.setOperators(A.M.handle)
initial_solver.setFromOptions()
initial_solver.setUp()

# Solve problem
solution = Function(P)
with b.dat.vec_ro as b_vec, solution.dat.vec as sol_vec:
  initial_solver.solve(b_vec,sol_vec)
```

Listing A.4: GMSH geometry file for Listing A.2

```
Point(1) = {0, 0, 0, 1.0};
Point(2) = {1, 0, 0, 1.0};
Point(3) = {1, 1, 0, 1.0};
Point(4) = {0, 1, 0, 1.0};
Point(5) = {4/9, 4/9, 0, 1.0};
Point(6) = {5/9, 4/9, 0, 1.0};
Point(7) = {5/9, 5/9, 0, 1.0};
Point(8) = {4/9, 5/9, 0, 1.0};
Line(1) = {1, 2};
Line(2) = {2, 3};
Line(3) = {3, 4};
Line(4) = {4, 1};
Line(5) = {5, 6};
Line(6) = {6, 7};
Line(7) = {7, 8};
Line(8) = {8, 5};
Line Loop(9) = {4, 1, 2, 3};
Line Loop(10) = {8, 5, 6, 7};
Plane Surface(11) = {9, 10};
Physical Line(12) = {4};
Physical Line(13) = {1};
Physical Line(14) = {2};
Physical Line(15) = {3};
Physical Line(16) = {7};
Physical Line(17) = {6};
Physical Line(18) = {5};
Physical Line(19) = {8};
Physical Surface(20) = {11};
```

Listing A.5: Semi-smooth (VI - SS) method

```
# Create TAO object
ss_solver = PETSc.TAO().create(PETSc.COMM_WORLD)
ss_solver.setOptionsPrefix("ss_")

# Semi-smooth call-backs
def ss_formJac(tao, petsc_x, petsc_J, petsc_JP, A=None, a=None, bcs=None):
  A = assemble(a, bcs=bcs, tensor=A)
  A.M._force_evaluation()
def ss_formFunc(tao, petsc_x, petsc_g, A=None, a=None, b=None, bcs=None):
  A = assemble(a, bcs=bcs, tensor=A)
  with b.dat.vec as b_vec:
    A.M.handle.mult(petsc_x, petsc_g)
    petsc_g.axpy(-1.0,b_vec)

# Setup
ss_con = Function(solution.function_space())
with ss_con.dat.vec as con_vec, lb.dat.vec as lb_vec, ub.dat.vec as ub_vec:
  ss_solver.setConstraints(ss_formFunc, con_vec, kargs={'A':A,'a':a,'b':b,'bcs':bcs})
  ss_solver.setJacobian(ss_formJac,A.M.handle, kargs={'A':A,'a':a,'bcs':bcs})
  ss_solver.setType(PETSc.TAO.Type.SSFLS) # can also be ASFLS/SSILS/ASILS
  ss_solver.setVariableBounds(lb_vec,ub_vec)
  ss_solver.setFromOptions()

# Solve problem
def viss():
  with solution.dat.vec as sol_vec:
    ss_solver.solve(sol_vec)
```

Listing A.6: Reduced-space active set (VI - RS) method

```
# Create SNES object
rs_solver = PETSc.SNES().create(PETSc.COMM_WORLD)
rs_solver.setOptionsPrefix("rs_")

# Reduced-space active-set call-backs
def rs_formJac(snes, petsc_x, petsc_J, petsc_JP):
  pass
def rs_formFunc(snes, petsc_x, petsc_g, A=None, b=None):
  with b.dat.vec as b_vec:
    A.M.handle.mult(petsc_x, petsc_g)
    petsc_g.axpy(-1.0,b_vec)
rs_con = Function(solution.function_space())

```

```
# Solve problem
def virs():
  with solution.dat.vec as sol_vec, lb.dat.vec as lb_vec, ub.dat.vec as ub_vec:
    with rs_con.dat.vec as con_vec:
      rs_solver.setFunction(rs_formFunc, con_vec, kargs={'A':A,'b':b})
    rs_solver.setJacobian(rs_formJac,A.M.handle)
    rs_solver.setType(PETSc.SNES.Type.VINEWTONRSLS)
    rs_solver.setVariableBounds(lb_vec,ub_vec)
    rs_solver.setFromOptions()
    rs_solver.solve(None,sol_vec)
    rs_solver.reset()
```

Listing A.7: Trust region Newton (QP - TRON) method

```
# Create TAO object
tron_solver = PETSc.TAO().create(PETSc.COMM_WORLD)
tron_solver.setOptionsPrefix("tron_")

# TRON call-backs
def tron_formHess(tao, petsc_x, petsc_H, petsc_HP):
  pass
def tron_formObjGrad(tao, petsc_x, petsc_g, A=None, b=None):
  with b.dat.vec_ro as b_vec:
    A.M.handle.mult(petsc_x, petsc_g)
    xtHx = petsc_x.dot(petsc_g)
    xtf = petsc_x.dot(b_vec)
    petsc_g.axpy(-1.0,b_vec)
    return 0.5*xtHx - xtf

# Setup
with lb.dat.vec as lb_vec, ub.dat.vec as ub_vec:
  tron_solver.setVariableBounds(lb_vec,ub_vec)
tron_solver.setObjectiveGradient(tron_formObjGrad, kargs={'A':A,'b':b})
tron_solver.setHessian(tron_formHess,A.M.handle)
tron_solver.setType(PETSc.TAO.Type.TRON)
tron_solver.setFromOptions()

# Solve problem
def qptron():
  with solution.dat.vec as sol_vec:
    tron_solver.solve(sol_vec)
```

Appendix B. Solution strategy for the Darcy equation

B.1 WEAK FORMULATION

Let $\mathbf{w}(\mathbf{x})$ and $q(\mathbf{x})$ represent the weighting functions for velocity and pressure respectively. The relevant function spaces read as follows:

$$\mathcal{V} := \left\{ \mathbf{v} \in (L_2(\Omega))^d \;\middle|\; \mathrm{div}[\mathbf{v}] \in L_2(\Omega),\ \mathbf{v} \cdot \widehat{\mathbf{n}} = v_n \text{ on } \Gamma^v \right\}, \tag{B.1.1a}$$

$$\mathcal{W} := \left\{ \mathbf{w} \in (L_2(\Omega))^d \;\middle|\; \mathrm{div}[\mathbf{w}] \in L_2(\Omega),\ \mathbf{w} \cdot \widehat{\mathbf{n}} = 0 \text{ on } \Gamma^v \right\}, \quad \text{and} \tag{B.1.1b}$$

$$\mathcal{P} := L_2(\Omega), \tag{B.1.1c}$$

where $L_2(\Omega)$ is the space of square integrable functions. The WF under the classical mixed formulation for the Darcy equations (4.6.1a) through (4.6.1d) reads: Find $\mathbf{v}(\mathbf{x}) \in \mathcal{V}$ and $p(\mathbf{x}) \in \mathcal{P}$ such that we have:

$$\mathcal{B}(\mathbf{w}, q; \mathbf{v}, p) = \mathcal{L}(\mathbf{w}, q) \quad \forall \mathbf{w}(\mathbf{x}) \in \mathcal{W},\ q(\mathbf{x}) \in \mathcal{P}, \tag{B.1.2}$$

where the bilinear form and linear functional are:

$$\mathcal{B}(\mathbf{w}, q; \mathbf{v}, p) := \left(\mathbf{w}(\mathbf{x});\ \frac{\mu(c(\mathbf{x}))}{k(\mathbf{x})} \mathbf{v}(\mathbf{x}) \right)_{\Omega} - \left(\mathrm{div}[\mathbf{w}(\mathbf{x})];\ p(\mathbf{x})\right)_{\Omega} - \left(q(\mathbf{x});\ \mathrm{div}[\mathbf{w}(\mathbf{x})]\right)_{\Omega} \quad \text{and} \tag{B.1.3}$$

$$\mathcal{L}(\mathbf{w}, q) := \left(\mathbf{w}(\mathbf{x});\ \rho \mathbf{b}(\mathbf{x})\right)_{\Omega} - \left(\mathbf{w}(\mathbf{x}) \cdot \widehat{\mathbf{n}}(\mathbf{x});\ p^{\mathrm{P}} \right)_{\Gamma^p}. \tag{B.1.4}$$

The lowest order Raviart-Thomas (RT0) space [143] is employed because it ensures element-wise mass conservation. To map the RT0 element onto quadrilateral and extruded hexahedrons, contravariant Piola mapping is used (see [146, 17] for further details). The discrete formulations may be assembled into the following block format:

$$\begin{pmatrix} \boldsymbol{K}_{vv} & \boldsymbol{K}_{vp} \\ \boldsymbol{K}_{pv} & \boldsymbol{K}_{pp} \end{pmatrix} \begin{pmatrix} \boldsymbol{v} \\ \boldsymbol{p} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}_v \\ \boldsymbol{f}_p \end{pmatrix}, \tag{B.1.5}$$

where the terms in equation (B.1.3) respectively correspond to $\boldsymbol{K}_{vv}$, $\boldsymbol{K}_{vp}$, and $\boldsymbol{K}_{pv}$, and equation (B.1.4) corresponds to $\boldsymbol{f}_v$. It should be noted that $\boldsymbol{K}_{pp}$ and $\boldsymbol{f}_p$ are a zero matrix and zero vector, respectively.

B.2 PRECONDITIONING METHODOLOGY

Equation (B.1.5) is a saddle-point system which is tricky to precondition effectively for large-scale problems. Several classes of iterative solvers and preconditioning strategies exist for these types of problems [16, 47, 125]. One could alternatively employ hybridization techniques [39] which introduces Lagrange multipliers to reduce the difficulty of solving such problems. In this dissertation, we employed a Schur complement approach to precondition the saddle-point system. Conceptually, the problem at hand is a 2×2 block matrix:

$$\boldsymbol{K} = \begin{pmatrix} \boldsymbol{K}_{vv} & \boldsymbol{K}_{vp} \\ \boldsymbol{K}_{pv} & \boldsymbol{0} \end{pmatrix}, \tag{B.2.1}$$

which admits a full factorization of

$$\boldsymbol{K} = \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{K}_{pv}\boldsymbol{K}_{vv}^{-1} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{K}_{vv} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{S} \end{pmatrix} \begin{pmatrix} \boldsymbol{I} & \boldsymbol{K}_{vv}^{-1}\boldsymbol{K}_{vp} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix}, \tag{B.2.2}$$

where $\boldsymbol{I}$ is the identity matrix and

$$\boldsymbol{S} = -\boldsymbol{K}_{pv}\boldsymbol{K}_{vv}^{-1}\boldsymbol{K}_{vp}, \tag{B.2.3}$$

is the Schur complement. The inverse can therefore be written as:

$$\boldsymbol{K}^{-1} = \begin{pmatrix} \boldsymbol{I} & -\boldsymbol{K}_{vv}^{-1}\boldsymbol{K}_{vp} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{K}_{vv}^{-1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{S}^{-1} \end{pmatrix} \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ -\boldsymbol{K}_{pv}\boldsymbol{K}_{vv}^{-1} & \boldsymbol{I} \end{pmatrix}. \tag{B.2.4}$$

The task at hand is to approximate $\boldsymbol{K}_{vv}^{-1}$ and $\boldsymbol{S}^{-1}$. Since $\boldsymbol{K}_{vv}$ is a mass matrix for the Darcy equation, we can invert it using the ILU(0) (incomplete lower upper) solver. We note that the Schur complement is spectrally a Laplacian, so we can employ a diagonal mass-lumping of $\boldsymbol{K}_{vv}$ to give a good approximation to $\boldsymbol{K}_{vv}^{-1}$. That is, we can use

$$\boldsymbol{S}_p = -\boldsymbol{K}_{pv}\text{diag}\left(\boldsymbol{K}_{vv}\right)^{-1}\boldsymbol{K}_{vp}, \tag{B.2.5}$$

to precondition the inner solver inverting $\boldsymbol{S}$. For this block we employ the multi-grid V-cycle on $\boldsymbol{S}_p$ using the Trilinos ML package ([147]). These blocks are symmetric and positive-definite so one could employ the CG solvers to obtain the inverses. When the inverses are obtained, only a single sweep of flexible GMRES is needed to obtain the full solution. However, instead of individually solving for $\boldsymbol{K}_{vv}^{-1}$ and $\boldsymbol{S}_p$, we could alternatively apply a single sweep of ILU(0) and multi-grid, respectively, and rely on GMRES to solve the entire block system. By providing less accurate approximations of the inner individual blocks, the number of GMRES iterations for the overall system increases but the numerical accuracy remains the same. We have found that this methodology is computationally less expensive and more practical for large-scale computations. One could alternatively employ one of the factorizations (either lower or upper) to decrease the computational cost associated with setting up the preconditioner. Below are some necessary PETSc command-line options for the described Schur complement approach.

Listing B.1: PETSc solver options for the Schur complement approach

```
parameters = {
  # Outer solver
  "ksp_type": "gmres",

  # Schur complement with full factorization
  "pc_type": "fieldsplit",
  "pc_fieldsplit_type": "schur",
  "pc_fieldsplit_schur_fact_type": "full",

  # Diagonal mass lumping
  "pc_fieldsplit_schur_precondition": "selfp",

  # Single sweep of ILU(0) for the mass matrix
  "fieldsplit_0_ksp_type": "preonly",
  "fieldsplit_0_pc_type": "ilu",

  # Single sweep of multi-grid for the Schur complement
  "fieldsplit_1_ksp_type": "preonly",
  "fieldsplit_1_pc_type": "ml"
}
```