A Study of Several Applications of Parallel Computing in the Sciences Using PETSC

Nicholas Stegmeier
South Dakota State University

Follow this and additional works at: https://openprairie.sdstate.edu/etd

Part of the Fluid Dynamics Commons, Mathematics Commons, and the Mechanical Engineering Commons

Recommended Citation
https://openprairie.sdstate.edu/etd/3147

This Thesis - Open Access is brought to you for free and open access by Open PRAIRIE: Open Public Research Access Institutional Repository and Information Exchange. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of Open PRAIRIE: Open Public Research Access Institutional Repository and Information Exchange. For more information, please contact michael.biondo@sdstate.edu.
A STUDY OF SEVERAL APPLICATIONS OF PARALLEL COMPUTING IN THE SCIENCES USING PETSC

BY

NICHOLAS STEGMEIER

A thesis submitted in partial fulfilment of the requirements for the

Master of Science

Major in Mathematics

South Dakota State University

2019
A STUDY OF SEVERAL APPLICATIONS OF PARALLEL COMPUTING IN THE
SCIENCES USING PETSC

NICHOLAS STEGMEIER

This thesis is approved as a creditable and independent investigation by a candidate for the Master of Science in Mathematics degree and is acceptable for meeting the thesis requirements for this degree. Acceptance of this does not imply that the conclusions reached by the candidates are necessarily the conclusions of the major department.

Jung-Han Kimn, Ph.D.
Thesis Advisor

Kurt Cogswell, Ph.D.
Head, Department of Mathematics and Statistics

Dean, Graduate School
This thesis is dedicated to my father, whose ingenuity I greatly admire and have tried to match; to my mother, who has always cared for my success; and to the countless scientists of history who have inspired me with their passion to understand the natural world.

“We must not believe those, who today, with philosophical bearing and deliberative tone, prophesy the fall of culture and accept the ignorabimus. For us there is no ignorabimus, and in my opinion none whatever in natural science. In opposition to the foolish ignorabimus our slogan shall be: Wir müssen wissen - wir werden wissen. (‘We must know - we will know.’)”

David Hilbert, 1930
ACKNOWLEDGEMENTS

I would like to acknowledge the tireless support of Dr. Jung-Han Kimn, Dr. Jeffrey Doom, and Dr. Nathan McClanahan, all of whom have generously poured into my education and development. In addition, I would like to acknowledge the support I received from Los Alamos National Laboratory (LANL), particularly from Dr. Marc Charest and staff scientists from CCS-7. The development of this thesis was only possible because of the many skills I learned while interning at LANL.
CONTENTS

ABBREVIATIONS ............................................................... viii

LIST OF FIGURES ............................................................. ix

LIST OF TABLES ............................................................... x

ABSTRACT ................................................................. xi

1 INTRODUCTION ............................................................ 1
  1.1 OVERVIEW OF PETSC .................................................. 3
  1.2 ORGANIZATION ......................................................... 4

2 SIMULATION OF IMPINGING JETS .................................... 5
  2.1 GOVERNING EQUATIONS ............................................... 6
  2.2 NUMERICAL METHODS ............................................... 7
  2.3 PARALLEL IMPLEMENTATION ....................................... 8
  2.4 IMPINGING JET SIMULATION PARAMETERS ...................... 13
  2.5 SIMULATION RESULTS ............................................... 14
  2.6 IMPINGING JETS FUTURE WORK ................................... 19

3 SIMULATION OF BIOFILMS USING THE MODIFIED CAHN-HILLIARD
  EQUATION ............................................................... 20
  3.1 MATHEMATICAL FORMULATION ................................... 21
  3.2 MOMENTUM TRANSPORT AND CONTINUITY ....................... 21
3.2.1 THE CAHN-HILLIARD EQUATION AS THE NETWORK TRANSPORT EQUATION ........................ 22
3.2.2 THE NUTRIENT TRANSPORT EQUATIONS ........................................ 23
3.2.3 THE CONSTITUTUVE EQUATIONS .................................................... 23
3.2.4 NONDIMENSIONALIZATION .......................................................... 24
3.3 NUMERICAL METHODS ................................................................. 25
3.4 PARALLEL IMPLEMENTATION ....................................................... 28
3.5 SIMULATION RESULTS ................................................................. 28
  3.5.1 2D FLUID FLOW WITH BIOFILM PROTRUSIONS ...................... 30
  3.5.2 BIOFILM DROPLET VALIDATION CASE ................................. 31
  3.5.3 3D FLOW AROUND A BIOFILM ............................................. 34
3.6 BIOFILMS FUTURE WORK ........................................................... 34

4  SIMULATION OF THE ALTERNATING CURRENT OPTIMAL POWER FLOW PROBLEM .................................................. 35
  4.1 FORMULATION OF ACOPF ....................................................... 37
  4.2 PRIMAL DUAL INTERIOR POINT METHOD .................................. 39
  4.3 PARALLEL IMPLEMENTATION .................................................... 40
  4.4 SIMULATION PARAMETERS ....................................................... 44
  4.5 PARALLEL PERFORMANCE ....................................................... 44
  4.6 ACOPF FUTURE WORK ........................................................... 48

5  CONCLUSION .................................................................................. 49
# LIST OF FIGURES

1. Hierarchy of objects in PETSc. ............................................. 4
2. An example computational domain when $N_x=4$, $N_y=4$. .......... 8
3. DMDA partitioning of mesh for two processors, with and without ghost points. ..................................................... 9
4. Velocity magnitude (left) and vorticity magnitude (right) for the 2D Taylor-Green vortex problem. ................................. 13
5. Strong scaling comparison of PETSc implementation versus original MPI implementation. .................................................. 13
6. Boundary conditions for the impinging jet problem. ................. 14
7. Perspective view of the scalar concentration for the fully developed jet at nondimensional time $t = 156$. ............................... 15
8. Streamwise instantaneous cross sections of the scalar concentration, temperature, velocity magnitude, and vorticity magnitude at nondimensional time $t = 156$. ................................................. 16
9. Centerline streamwise cross section of the mean scalar concentration. ................................................................. 16
10. Centerline streamwise cross section of the scalar concentration variance. .......................................................... 16
11. Centerline streamwise cross section of the mean temperature. .... 17
12. Centerline streamwise cross section of the temperature variance. . 17
13. Impingement plane cross section of the mean (left) and variance (right) of the scalar concentration. .................................... 18
14 Impingement plane cross section of the mean (left) and variance (right) of the temperature.

15 Strong scaling raw simulation times compared to an ideal scaling reference line.

16 Strong scaling speedup compared to the ideal case.

17 A cartoon of the biofilm in 2D.

18 A 256x128 simulation of two viscous biofilm droplets immersed in a less viscous medium.

19 A 256x256x128 simulation of two viscous biofilm droplets suspended in a less viscous medium.

20 A 256x256x128 simulation of two biofilm droplets, $t = 2$, with slices to visualize the inner concentration.

21 Streamlines over a bumpy biofilm surface.

22 Overview of the algorithm and implementation.

23 Execution time versus number of CPU cores in three different case systems.

24 Parallel speedup of ACOPF code in the 118 bus (blue), 300 bus (red), and 1354 bus (green) test systems. Significant performance variation is seen due to varying parallel decomposition and subdomain construction with the ASM preconditioner.

25 Affect of ASM overlap on PDIPM residual and number of KSP iterations per outer step.

26 Affect of ASM overlap on ACOPF runtime.
# LIST OF TABLES

1. PETSc Pressure Iterations .................................................. 12
2. JS Pressure Iterations ....................................................... 12
3. Impinging jet conditions. ................................................... 15
4. List of parameters values used for the simulations. ............... 30
ABSTRACT

A STUDY OF SEVERAL APPLICATIONS OF PARALLEL COMPUTING IN THE SCIENCES USING PETSC

NICHOLAS STEGMEIER

2019

The importance of computing in the natural sciences continues to grow as scientists strive to analyze complex phenomena. The dynamics of turbulence, astrophysics simulations, and climate change are just a few examples where computing is critical. These problems are computationally intractable on all computing platforms except supercomputers, necessitating the continued development of efficient algorithms and methodologies in parallel computing. This thesis investigates the use of parallel computing and mathematical modeling in the natural sciences through several applications, namely computational fluid dynamics for impinging jets in mechanical engineering, simulation of biofilms in an aqueous environment in mathematical biology, and the solution of the alternating current optimal power flow problem in electrical engineering. In this study, the Portable, Extensible Toolkit for Scientific Computing (PETSc) is utilized in the design of the numerical methods and parallel implementations.
1 INTRODUCTION

Supernova, jet engines, nuclear fission and fusion: the problem connecting these disparate phenomena is the difficulty, expense, or danger of collecting useful data. Modeling and simulation is a solution that has the potential to revolutionize the way scientific questions are asked and answered. The explosion of a star, arguably the most violent event in the universe, can be safely analyzed in a simulation. However, there are major challenges to overcome to realize this revolution: many questions worth answering require a supercomputer, and the mathematics and methodology of modeling physical phenomena is far from complete. In a broad sense, high performance computing (HPC) is the study of computer science, numerical analysis, and mathematical modeling needed to solve the challenging problems presented to modern science, like those listed above. As computing plays an ever more important role in science, the study of high performance (and parallel) computing likewise grows in importance.

Applications of high performance computing today include predicting climate dynamics years in the future [14], simulating the evolution of the early universe and subsequent galactic development [28], and decoding immensely complicated genetic patterns [33], to name a few. Many others have been documented by the Department of Energy [52]. National laboratories, universities, and financial institutions all own supercomputers of varying size and purpose with dedicated teams of engineers and scientists to put them to use. With the broad usage of HPC, research is active across disciplines to improve the hardware, software, and mathematical modeling so that more accurate, more efficient, and more robust systems can be developed. As a showcase example, the new 200 petaflop supercomputer Summit, located at Oak Ridge National Laboratory, features a hybrid architecture of CPUs and GPUs [34]. The heterogeneous architecture on Summit pushes developers to adapt their codes and algorithms to utilize each component efficiently. In response, software is being developed at Sandia National
Laboratories [5] and Los Alamos National Laboratory [4] to enable high productivity development even on heterogeneous platforms. Summit provides 18,688 nodes, and using a large fraction of these nodes in parallel poses a severe challenge to even the most performant scientific codes. Mathematicians are developing algorithms that can operate in asynchronous fashion [51], pointing a way towards performance at even the most extreme scales, as required by exascale computing [52].

In the context of these exciting challenges, this thesis investigates the use cases, methodologies, and mathematical modeling involved in three applications of high performance computing: direct numerical simulation of impinging jets using the Navier-Stokes equations; simulation of biofilms in an aqueous environment using the Modified Cahn-Hilliard equation coupled to the incompressible Navier-Stokes equations; and solving the Alternating Current Optimal Power Flow (ACOPF) problem for the optimal dispatch of power in an electrical grid.

The first two examples involve the Navier-Stokes equations, a coupled system of 2nd order nonlinear partial differential equations (PDEs). A variety of numerical methods exist to solve these PDEs, the most common being the finite difference, finite volume, and finite element methods. Typically, these methods approximate the continuous differential operators, leading to a discrete system of linear algebraic equations of the form $Ax = b$. For low order methods with finely resolved meshes, the matrix $A$ is sparse. Thus $Ax = b$ may be solved with parallel, iterative linear solvers, discussed in more detail later. The finite volume and finite difference methods with structured meshes are used for the problems considered in this thesis. A distinguishing feature of the biofilm problem, as compared to the impinging jets application, is the coupling of the 4th order, nonlinear Cahn-Hilliard equation, which adds substantial complexity to the discretization.

The ACOPF problem is a nonlinear optimization problem and requires a different numerical approach. The electrical power grid is represented as an unstructured grid of nodes. A cost function and system of nonlinear algebraic constraints is generated by
enforcing the appropriate relationships for voltage, power, and line limits between the nodes in the electrical grid. The goal of investigating these disparate applications is to give a broader perspective on the implementation and use of parallel computing for solving mathematical problems motivated by real-life situations.

1.1 OVERVIEW OF PETSC

This thesis uses PETSc for the parallel implementation of several application problems. PETSc is a software library for parallel computing developed at Argonne National Laboratory. Implementing efficient algorithms for modern scientific applications requires the use of a swath of concepts in computer science and numerical analysis, and it would be impractical and ineffective for one person to implement everything. PETSc provides data structures and routines for creating efficient, scalable solution methods for problems that can be modeled by partial differential equations [2]. Since very many problems in science can be modeled by PDEs, PETSc has been used frequently in the academic community [2]. Some of the basic data structures in PETSc include matrices, vectors, and index sets. Higher level objects like unstructured mesh utilities, iterative linear solvers, and preconditioners take the basic data structures as inputs. Parallelism of data structures and numerical routines is accomplished with the Message Passing Interface (MPI), which PETSc uses behind the scenes. MPI is a message passing library for parallel computing based on the consensus and input of industry leaders in high performance computing [15]. Programming a scientific application from scratch using MPI is a difficult endeavor, and PETSc aims to hide most of that effort from the scientist. For an accomplished PETSc programmer, few if any calls to MPI are required to design a parallel program. The hierarchy of objects in PETSc is shown in Figure 1, and the specific uses of PETSc in this thesis are detailed in later sections.
1.2 ORGANIZATION

The remainder of this thesis is organized as follows. Section 2 describes the simulation of impinging jets, including a discussion of direct numerical simulation for problems in computational fluid dynamics and the use of PETSc for structured mesh numerical PDE problems. Section 3 describes the simulation of biofilms in an aqueous environment using the modified Cahn-Hilliard equation. Whereas the impinging jet problem uses an implementation of the compressible Navier-Stokes equations, the biofilm problem requires only the incompressible equations, but the complexity is increased substantially through the coupling of the Cahn-Hilliard equation, a nonlinear 4th-order PDE. Section 4 describes the use of PETSc for a nonlinear optimization problem from electrical engineering, the ACOPF problem. This problem requires the use of an unstructured network object from PETSc. The affect of PETSc preconditioner options on the convergence of the nonlinear optimization problem is investigated. Finally, overall conclusions are given in section 5, and the appendix provides a detailed account of the discretization for the biofilm problem.
2 SIMULATION OF IMPINGING JETS

The starting point of this section is an existing computational fluid dynamics (CFD) research code which uses an implicit, parallelized Jacobi solver (JS) [10]. We refer to the existing implementation as JS. The JS algorithm was replaced with routines and data structures from PETSc. The performance of the modified code is analyzed, and large scale simulations are performed for an impinging jet in crossflow.

The impinging jet technique for cooling the blades of gas turbines was introduced in the early 1960s [31]. Jet impingement is one of the successful and broadly used methods for exchanging mass and energy where a jet of fluid impinges against a surface and heat transfer is required. It is believed to give three to four times higher heat transfer coefficient compared to the ordinary convection cooling jet [59]. The use of impinging jets is widely increasing due to the high heat and mass transfer as compared to other methods. This technique has been used in industries for the production of glass, processing of steel, electrical cooling equipment, and the cooling of turbine blades. Although the technique involves a simple geometry involving stagnation and adverse pressure zone, most of the applications possess complex flow characteristics.

CFD methods such as large eddy simulations (LES) and Direct Numerical Simulation (DNS) have assisted in analyzing flow and heat transfer characteristics of jets. An overview is given by Zuckerman and Lior which covers impinging jet flow structure and physics, correlations, and a comparison of numerical approaches [59]. They concluded that Reynolds-Averaged Navier-Stokes (RANS) simulations sometimes have large errors in the predicted heat transfer characteristics, while LES and DNS simulations can predict the flow and heat transfer to relatively high accuracy. However, they did not discuss impinging jets with crossflow in detail. Rizk and Menon studied impinging jets using LES and worked on two-jet impingement CFD to find the dynamical interaction of the flow structures[44]. Heat transfer for an impinging circular jet with and without
crossflow was experimentally studied by Goldstein and Bebblock [17]. Lei Wang et al. studied an inclined impinging jet in crossflow experimentally and found that crossflow diminishes the peak Nusselt number [50]. The flow characteristics and coherent structures of free jets in crossflow are reviewed by Mahesh [29]. While significant research has been conducted experimentally and numerically for free and impinging jets with and without crossflow, the present literature would benefit from further high fidelity DNS simulations of impinging jets in crossflow. In this section, DNS results are presented of an impinging jet in crossflow with constant temperature boundary conditions at the impingement plate.

2.1 GOVERNING EQUATIONS

The governing equations are the unsteady, compressible Navier–Stokes equations:

\[
\frac{\partial \rho d}{\partial t} + \frac{\partial \rho u_j d}{\partial x_j} = 0, \tag{1}
\]

\[
\frac{\partial \rho dY_k d}{\partial t} + \frac{\partial \rho Y_k d u_j d}{\partial x_j} = \frac{\partial}{\partial x_j} \left( \rho d D_k d \frac{\partial Y_k d}{\partial x_j} \right), \tag{2}
\]

\[
\frac{\partial \rho d u_i d}{\partial t} + \frac{\partial \rho d u_i d u_j d}{\partial x_j} = -\frac{\partial p d}{\partial x_i} + \frac{\partial \tau_{ij} d}{\partial x_j} \tag{3}
\]

\[
\frac{\partial \rho d E d}{\partial t} + \frac{\partial}{\partial x_j} \left( \rho d E d + p d \right) u_j d = \frac{\partial \tau_{ij} d u_i d}{\partial x_j} + \frac{\partial}{\partial x_j} \left( \mu d P r \frac{\partial T d}{\partial x_j} \right), \tag{4}
\]

\[
p d = \rho d R d T d^4 = \rho d \frac{R_a}{W d} T d. \tag{5}
\]

The superscript ‘\(d\)’ denotes the dimensional value. The pressure is non–dimensionalized using an incompressible scaling motivated by Thompson [48] which is defined as:

\[
p = \frac{\rho d - \rho r}{\rho r u_r^2} \text{ where the subscript ‘\(r\)’ denotes the reference variable. Therefore, the}
\]
non–dimensional governing equations from Doom et al. [9] are:

\[
\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_j}{\partial x_j} = 0, \quad (6)
\]

\[
\frac{\partial \rho Y_k}{\partial t} + \frac{\partial \rho Y_k u_j}{\partial x_j} = \frac{1}{Re Sc_k} \frac{\partial}{\partial x_j} \left( \mu \frac{\partial Y_k}{\partial x_j} \right), \quad (7)
\]

\[
\frac{\partial \rho u_i}{\partial t} + \frac{\partial \rho u_i u_j}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{1}{Re} \frac{\partial \tau_{ij}}{\partial x_j}, \quad (8)
\]

\[
M_r^2 \left[ \frac{\partial}{\partial t} \left( p + \gamma - \frac{1}{2} \rho u_i u_i \right) + \frac{\partial}{\partial x_j} \left( \gamma p + \gamma - \frac{1}{2} \rho u_i u_i \right) u_j \right] + \frac{\partial u_j}{\partial x_j} = \frac{(\gamma - 1)M_r^2 \partial \tau_{ij} u_i}{Re} + \frac{1}{Re Pr} \frac{\partial}{\partial x_j} \left( \mu \frac{\partial T}{\partial x_j} \right), \quad (9)
\]

\[
\rho T W = \gamma M_r^2 p + 1. \quad (10)
\]

Here, \( \rho, T, p, u_i \) and \( Y_k \) denote non–dimensional density, temperature, pressure, velocities and mass fraction of ‘\( k \)’ species respectively. The viscous stress tensor

\[
\tau_{ij} = \mu \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3} \frac{\partial u_k}{\partial x_k} \delta_{ij} \right). \quad Sc_k \text{ is the Schmidt number for the } k^{th} \text{ species, } Pr \text{ is the Prandtl number, and } Re \text{ is the Reynolds number. } W \text{ is the mean molecular weight of the mixture.}
\]

2.2 NUMERICAL METHODS

The numerical algorithm is discussed in detail by Doom et al. [9]. The algorithm is fully implicit, spatially non–dissipative and second order in time and space. A pressure–correction method is used to enforce velocity divergence obtained from the energy equation. The discrete equations discretely conserve kinetic energy in the incompressible, non–reacting, inviscid limit. These features make the algorithm stable and accurate at high Reynolds numbers, and efficient at both low and finite Mach numbers. These features are attractive for DNS of compressible flows.
2.3 PARALLEL IMPLEMENTATION

Principally, PETSc replaces the original MPI implementation in the JS code and is used to solve the linear systems resulting from the Navier-Stokes equations. This requires placing the linear system coefficients into PETSc matrices and vectors, which can then be used with PETSc Krylov Subspace (KSP) routines. There are several challenges to this approach that are closely related: (1) storing the data associated with each nodal point in the grid, (2) communicating these data between processors when necessary, and (3) relating the 3D mesh of nodal points with the resulting matrix of coefficients in parallel and in an extensible, user-friendly manner. PETSc provides a set of data management routines for distributed arrays (DMDA) that addresses all three challenges.

To provide a concrete example of the use of PETSc, DMDA, and MPI, consider a computational domain of \( \text{Nx}=4 \) by \( \text{Ny}=4 \) nodes. PETSc’s DMDA object must be provided with the number of nodal points in each direction, the stencil type and width, and the boundary conditions in each direction. Given this information, PETSc determines an efficient way of partitioning the mesh among processors. The DMDA object is initialized using \text{DMDACreate2D}().
Listing 1: Initializing a 2D DMDA object in PETSC.

```
// Inputs:
// PETSC_COMM_WORLD  PETSc communicator
// bx, by  boundary types
// stype  stencil type
// Nx,Ny  number of nodes
// dof  degrees of freedom
// sw  stencil width
// da  DMDA object

DMDACreate2d(PETSC_COMM_WORLD, bx, by, stype, Nx, Ny, PETSC_DECIDE, PETSC_DECIDE, dof, sw, NULL, NULL, &da);
```

The details of the resulting decomposition will depend on the number of nodes and number of processors in use. For a two processor case, DMDA might decompose the domain as follows:

![Diagram of mesh partitioning with and without ghost points.]

Figure 3: DMDA partitioning of mesh for two processors, with and without ghost points.

One of the more useful features of DMDA is the ability to associate matrices and vectors with the domain decomposition. A matrix $A$, solution vector $x$ and right-hand side
vector \( b \) can be associated with the \( \text{da} \) object created in Listing 1.

Listing 2: Associating matrices and vectors with DMDA.

```c
// Create PETSc matrix A and vectors x, b
DMCreateMatrix(da,&A);
DMCreateGlobalVector(da,&x);
DMCreateGlobalVector(da,&b);
```

Normally, the user must manually decompose and parallelize the computational domain and then parallelize matrices and vectors in a manner that is conformal with the domain parallelization. In Listing 2, this process is handled automatically. Furthermore, when constructing the matrix it is usually necessary to carefully map the domain indices to matrix rows and columns. PETSc offers the `MatStencil` data structure as way to automatically compute these index transformations. For example, at node \( (i,j) \) in the interior of the domain, a matrix row corresponding to the discrete 2D Laplacian could be entered as follows:

Listing 3: Constructing the matrix with DMDA.

```c
// Value and index data structures
PetscReal vals[5];
MatStencil rows, cols[5];
```

```c
// specify matrix entries
vals[0]=4;
```

```c
// specify matrix rows and columns
```
row.i = i; row.j = j;
col[0].i = i; col[0].j = j;
col[1].i = i-1; col[1].j = j;
col[2].i = i+1; col[2].j = j;
col[3].i = i; col[3].j = j-1;
col[4].i = i; col[4].j = j+1;

// set the matrix values and entries
MatSetValuesStencil(A, 1, &row, 5, col, vals, INSERT_VALUES);

Because matrix A was associated with the DMDA object da at initialization, PETSc is able to map the node indices \((i,j)\) to the corresponding matrix rows and columns. In addition, boundary conditions are more easily implemented using DMDA. For periodic boundary conditions, DMDA maps indices that are too large or small to the other edge of the domain. If, for example, \(i-1=-1\) in Listing 3, then DMDA would map this entry to the matrix column corresponding to \(i=nx-1\).

Listing 4: Creating PETSc KSP and PC objects.

// set the KSP method
KSPCreate(PETSC_COMM_WORLD,&ksp);
KSPSetType(ksp,KSPGMRES);

// set the PC
KSPGetPC(ksp,&pc);
PCSetType(pc,PCJACOBI);
KSPSetPC(ksp,pc);

PETSc was employed in such a way as to preserve as much of the original code as
possible. In practice, this means that the manual use of MPI was generally replaced with PETSc calls, but most other parts of the code were left alone. The one other area of the code that did see substantial modification was inside the time stepping loop, where the Navier-Stokes equations are solved. The Jacobi solver and convergence criteria were replaced with PETSc KSP routines and a preconditioner. By looping through the mesh subregion local to each processor and assigning matrix elements using PETSc DMDA, the structure of the time step loop is largely preserved, but numerical routines provided by PETSc KSP can easily be interchanged via command line or at the beginning of the program.

As a validation problem, the 2D Taylor-Green vortex problem was simulated for various Mach number and time discretization values and compared the average number of pressure-solver iterations required for convergence at each time step. This test was performed for both the original JS code and the modified PETSc code with a GMRES solver.

<table>
<thead>
<tr>
<th>Table 1: PETSc Pressure Iterations</th>
<th>Table 2: JS Pressure Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ma ( \Delta t ) ( 10^{-4} )</td>
<td>Ma ( \Delta t ) ( 10^{-4} )</td>
</tr>
<tr>
<td>( 10^{-1} ) 10 11 57</td>
<td>( 10^{-1} ) 8 13 131</td>
</tr>
<tr>
<td>( 10^{-2} ) 10 42 753</td>
<td>( 10^{-2} ) 10 76 21,164</td>
</tr>
<tr>
<td>( 10^{-3} ) 33 260 2,007</td>
<td>( 10^{-3} ) 66 2,887 69,382</td>
</tr>
<tr>
<td>( 10^{-4} ) 252 1,080 2,992</td>
<td>( 10^{-4} ) 2,729 65,141 29,308</td>
</tr>
</tbody>
</table>

For large Mach numbers and fine temporal discretizations, the PETSc GMRES code performed similarly to the original code. However, for numerically stiff problems with small Mach numbers and large time steps, an order of magnitude or more improvement is seen in the number of iterations. This implies that the PETSc code is more robust in a practical sense because it can solve numerically stiff problems in a reasonable amount of time.

Strong scaling data was collected for both the original and the modified code. The 2D Taylor-Green vortex problem with Mach number \( 10^{-3} \), Reynolds number 5,000, and...
time step $10^{-3}$ was used for the scaling test. In this physical regime, the original code and PETSc GMRES solver behave similarly in terms of iterations to solve the linear systems, so this scaling test roughly measures whether PETSc is using the computing hardware more efficiently. On the average, the PETSc implementation showed a $2X$ speedup relative to the original code.

Figure 5: Strong scaling comparison of PETSc implementation versus original MPI implementation.

2.4 IMPINGING JET SIMULATION PARAMETERS

As an engineering application for the improved CFD code using PETSc, simulation results are presented for a single impinging jet in crossflow. The jet is represented
numerically as a scalar concentration inflow with non-dimensional downward velocity of $u_{in} = 1.0$ at the top of a rectangular domain. A uniform crossflow of $u_{cross} = 0.1$ in the positive x-direction is initialized everywhere in the domain and enforced at the inflow boundary condition on the left. An outflow boundary condition is given at the right, and wall boundary conditions are enforced everywhere else.

![Diagram of boundary conditions for the impinging jet problem.](image)

Figure 6: Boundary conditions for the impinging jet problem.

The non-dimensionalization is identical to that described by Doom [10]. The reference pressure, temperature, density, dynamic viscosity, and velocity are, respectively, $p_r = 101,325 \text{ Pa}$, $T_r = 300 \text{ K}$, $\rho_r = 1.1768 \text{ kg/m}^3$, $\mu = 1.855 \times 10^{-5} \text{ Pa}\cdot\text{s}$, and $u_r = 34.72 \text{ m/s}$. With these reference quantities, the initial conditions for the jet simulation are given in Table 3, where $u_{jet}$ is the velocity of the jet, $u_{cross}$ is the crossflow velocity, $T_{jet}$ is the temperature of the jet, $T_{amb}$ is the temperature of the impingement plate and the ambient air, $D$ is the jet diameter, $H$ is the distance from jet to target, $p$ is the pressure, and $\rho$ is the density.

### 2.5 SIMULATION RESULTS

The impinging jet was simulated on a mesh of 400x100x200 with time step $dt = 0.001$ to a nondimensional time of $t = 215$. Instantaneous cross sections of the fully developed jet are shown in Figures 7-8.

The instantaneous flow field is highly unsteady, especially at the impingement plane and directly upstream of the jet. At the particular time shown in Figure 8, the
Table 3: Impinging jet conditions.

<table>
<thead>
<tr>
<th>var.</th>
<th>non-dim.</th>
<th>dim.</th>
<th>units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_{\text{jet}}$</td>
<td>1</td>
<td>34.72</td>
<td>m/s</td>
</tr>
<tr>
<td>$u_{\text{cross}}$</td>
<td>0.1</td>
<td>3.472</td>
<td>m/s</td>
</tr>
<tr>
<td>$T_{\text{jet}}$</td>
<td>1</td>
<td>300</td>
<td>K</td>
</tr>
<tr>
<td>$T_{\text{amb}}$</td>
<td>4</td>
<td>1200</td>
<td>K</td>
</tr>
<tr>
<td>$T_{\text{wall}}$</td>
<td>4</td>
<td>1200</td>
<td>K</td>
</tr>
<tr>
<td>$D$</td>
<td>1</td>
<td>0.0014</td>
<td>m</td>
</tr>
<tr>
<td>$H$</td>
<td>2</td>
<td>0.0028</td>
<td>m</td>
</tr>
<tr>
<td>$p$</td>
<td>0</td>
<td>101,325</td>
<td>Pa</td>
</tr>
<tr>
<td>$\rho$</td>
<td>1</td>
<td>1.1768</td>
<td>kg/m$^3$</td>
</tr>
</tbody>
</table>

Figure 7: Perspective view of the scalar concentration for the fully developed jet at nondimensional time $t = 156$. 
crossflow clearly forces the jet center slightly downstream. Although this feature is always present instantaneously, the jet impingement center is on the average shifted downstream. Impinged fluid that would, in the absence of crossflow, flow outwards radially from the jet impingement center is instead advected back towards and around the jet.

Mean flow field data was calculated for the fully developed jet with samples taken every 2,000 time steps for a total of 50 samples. The cross section shown in Figure 9 illustrates the mean scalar concentration of the jet and, by analogy, the mass transfer. The chosen crossflow $u_{\text{cross}} = 0.1$ shifts the jet center about $0.2D$ forward.

A plume upstream of the jet can be seen around $x/D = -2$ where the impinging fluid is curled upwards by the crossflow. This generates a dynamic flow region with high vorticity, which is reflected in the region of high variance for $-3 < x/D < 0$, as seen in Figure 10. This fluid is then pushed downstream towards and around the jet, contributing to a region of relatively low streamwise velocity directly upstream of the jet impingement.
The mean and variance of the temperature, shown in Figures 11 and 12 respectively, are similar to the scalar concentration. As the jet impinges onto the heated surface and moves laterally and downstream, the temperature varies from the initial 300K to around 700K directly downstream of the jet. Eventually, near the outflow boundary, the jet fluid nearly reaches the surrounding initial temperature of 1200K. Immediately upstream of the jet, the incoming crossflow slows, bifurcates, and travels around the jet. In this region ambient fluid is least affected by the cooling jet and the local average temperature remains near 1200K.

![Figure 11: Centerline streamwise cross section of the mean temperature.](image)

![Figure 12: Centerline streamwise cross section of the temperature variance.](image)

The region immediately after the impinging jet also has high scalar concentration and temperature variance, but for different reasons. In this region, $1 < x/D < 4$, the impinged fluid moves in the same direction as the crossflow, but at a higher velocity. This velocity gradient gives rise to shear forces and the jet fluid is sporadically curled up into the beginnings of vortices as it moves towards the outflow boundary. However, these vortical structures are quickly dispersed by the crossflow.

While the streamwise cross section gives a useful picture of the flow structures, the impingement plane is useful for analyzing the heat transfer characteristics. The present study uses constant temperature boundary conditions, while constant heat flux conditions
are planned for future work. In this study, a plate-parallel cross section of the flow is taken immediately above the impingement plane at $y/D = 1.9$. These cross sections again show the close relationship between the passive scalar and temperature field.

Figure 13: Impingement plane cross section of the mean (left) and variance (right) of the scalar concentration.

Figure 14: Impingement plane cross section of the mean (left) and variance (right) of the temperature.
2.6 IMPINGING JETS FUTURE WORK

A CFD implementation is presented using PETSc DMDA to handle parallel data communication and the solution of linear algebraic systems that result from discretizing the Navier-Stokes equations. This implementation is implicit, parallel and iterative. A previously implemented code was compared to the PETSc implementation in terms of parallel performance. The PETSc implementation was faster in all cases tested. As an application problem, we presented simulation results from a single impinging jet in crossflow. The instantaneous velocity, passive scalar distribution, temperature, and vorticity for the fully developed impinging jet are described. Time-averaged results for the fully developed flow were presented including the temperature and passive scalar distribution at the impingement plane. In the future, we plan to further analyze the single impinging jet case, particularly for heat flux boundary conditions, and extend the simulation to arrays of impinging jets in crossflow.
3 SIMULATION OF BIOFILMS USING THE MODIFIED CAHN-HILLIARD EQUATION

Biofilms are found throughout the world. They are common in both nature and in man-made environments. Biofilms are a collection of micro-organisms that adhere to a surface through a self-produced extracellular polymeric substance (EPS). This is a sticky, slimy substance that holds the biofilm in place. Biofilms cause massive losses around the world. They cost the U.S. alone billions of dollars every year. Biofilms contaminate food, water, and industrial equipment. They are also estimated to be the leading cause of chronic infections in the world [8]. Biofilms are not all bad though. They play a role in bioremediation, they are a safer, less toxic way to mine certain hard to obtain minerals, and they can be used as sealants through biomineralization [47, 36, 46].

Due to widespread nature of biofilms, the massive monetary losses caused by them, and the potential benefits from using them the last three decades have seen an increased desire to study them [41, 39, 38, 42]. There are models that are discrete where the biofilm is modeled using a cellular automaton approach[37] and there are models that are continuum based[1]. There are also models that are a combination of the two, where the cellular automaton approach is used for the biofilm and the continuum model is used for the nutrient substrate [40].

There are many more aspects of a model besides just discrete or continuum based. There are single fluid and multifluid models [7, 54]. There are models with multiple components to the fluid [54]. Varying the number of species of organisms in the biofilm can produce different results [1]. Biofilms have been treated as both elastic and visco-elastic fluids depending on the time scale used [54]. Experimental results support the use of a visco-elastic fluid model [27].

High fidelity simulation of biofilms is numerically challenging and computationally expensive. In addition to traditional concerns from computational fluid
dynamics such as ensuring a divergence free velocity field and coupling the pressure and velocity, the Cahn-Hilliard equation introduces a nonlinear, fourth order equation coupled to the momentum equation. This necessitates a larger stencil and more interprocessor communication. The discretized equations result in several sparse matrix systems of the form \( Ax = b \). Direct solution of these systems is often intractable due to their large size, so parallel iterative methods are pursued instead. PETSc is used to decompose the computational domain and iteratively solve the resulting linear systems in parallel. In particular, as in section 2, the DMDA abstraction layer in PETSc is used as an efficient interface for mapping the computational domain to the matrix. Previous work exists using parallel computational methods and variations of the Cahn-Hilliard equation. One of these includes work done by Zheng, Yang, Cai, and Keyes [56]. Their model differs in that they use the Cahn-Hilliard-Cook equation instead which has an additional term to model noise, in this case thermal fluctuations, in the system.

### 3.1 MATHEMATICAL FORMULATION

In this thesis, the one-fluid two-component model developed in [54, 53] is used. Define the following variables. Let the average velocity be \( \mathbf{v} \), the pressure be \( p \), the volume fraction of the polymer be \( \phi_n \), the volume fraction of the solvent be \( \phi_s \), and \( c \) be the nutrient concentration.

### 3.2 MOMENTUM TRANSPORT AND CONTINUITY

Incompressible flow is assumed, which results in the following continuity equations for momentum transport

\[
\nabla \cdot \mathbf{v} = 0, \quad (11)
\]

\[
\rho \frac{d\mathbf{v}}{dt} = \nabla \cdot (\phi_n \mathbf{\tau}_n + \phi_s \mathbf{\tau}_s) - [\nabla p + \gamma_1 k T \nabla \cdot (\nabla \phi_n \nabla \phi_n)], \quad (12)
\]

\]
where the density is a volume fraction averaged density \( \rho = \phi_n \rho_n + \phi_s \rho_s \) with \( \rho_n \) being the density of the polymer network, \( \rho_s \) is the density of solvent, \( \tau_n \) is the extra stress tensor for the network, \( \tau_s \) is the extra stress tensor for the solvent, \( k \) is the Boltzman constant, \( T \) is the temperature, and \( \gamma_1 \) measures the strength of the conformation entropy. A volume fraction averaged velocity field is given by \( v = \phi_n v_n + \phi_s v_s \), and a phase separation energy functional is used instead of the extended Flory-Huggin’s mixing free energy density for ease of use

\[
\frac{\gamma_1}{2} ||\nabla \phi_n||^2 + \gamma_2 kT \left( \phi_n^2 (1 - \phi_n)^2 \right)
\]

(13)

where \( \gamma_2 \) is the strength of the bulk free energy from the Flory-Huggin’s free energy density. The incompressible condition also implies that

\[
\phi_n + \phi_s = 1
\]

3.2.1 THE CAHN-HILLIARD EQUATION AS THE NETWORK TRANSPORT EQUATION

The polymer network transport equation is modeled using the singular or modified Cahn-Hilliard equation

\[
\frac{\partial \phi_n}{\partial t} + \nabla \cdot (\phi_n v) = \nabla \cdot \left( \lambda \phi_n \nabla \frac{\delta f}{\delta \phi} \right) + g_n.
\]

(14)

where \( \lambda \) is the mobility parameter and the network production rate is given by

\[
g_n = \epsilon \mu \phi_n \frac{c}{K_c + \epsilon'}
\]

(15)

where \( \epsilon \) is a scaling constant, \( \mu \) is maximum production rate, and \( K_c \) is the half-sturation constant. The Cahn-Hilliard equation has a polymer network volume fraction dependent
mobility. This equation is chosen as opposed to the standard Cahn-Hilliard equation as it has been shown that the modified Cahn-Hilliard equation is more appropriate to use when modeling the transport of the polymer network especially with production of polymer included [54].

### 3.2.2 THE NUTRIENT TRANSPORT EQUATIONS

The nutrient transport equation is given by

$$\frac{\partial}{\partial t} (\phi_s c) + \nabla \cdot (c \mathbf{v} \phi_s - D_s \phi_s \nabla c) = -g_c$$  \hspace{1cm} (16)

where $c$ is the nutrient concentration, $D_s$ is the diffusion coefficient for the nutrient substrate, and the nutrient consumption rate is given by

$$g_c = \phi_n A_c$$  \hspace{1cm} (17)

with $A$ being a constant.

### 3.2.3 THE CONSTITUITUVE EQUATIONS

The constituitive equations are as follows

$$\tau_n = 2\eta_n \mathbf{D}, \quad \tau_s = 2\eta_s \mathbf{D},$$  \hspace{1cm} (18)

where $\eta_n$ and $\eta_s$ are the viscosities of the polymer network and solvent. The rate of strain tensor, $\mathbf{D}$, is given by

$$\mathbf{D} = \frac{1}{2} [\nabla \mathbf{v} + \nabla \mathbf{v}^T].$$  \hspace{1cm} (19)

The velocities $\mathbf{v}_n$ and $\mathbf{v}_s$ have two parts to them. The first part is the convection due to the average velocity $\mathbf{v}$. For $\mathbf{v}_n$ the second part is an excessive flux due to the mixing of the polymer network and solvent. The polymer network excessive flux is defined as
proportional to the gradient of the variation of the free energy.

\[ \mathbf{v}_n^e = -\lambda \phi_n \nabla \frac{\delta f}{\delta \phi_n} \]  

(20)

The excessive flux for the solvent is due to the spatial inhomogeneity of the mixture and is defined as [54].

\[ \mathbf{v}_s^e = \lambda \phi_n \frac{\phi_s}{\phi_n} \nabla \frac{\delta f}{\delta \phi_n} \]  

(21)

The network velocity \( \mathbf{v}_n \) and solute velocity \( \mathbf{v}_s \) as follows

\[ \mathbf{v}_n = \mathbf{v} + \mathbf{v}_n^e \quad \mathbf{v}_s = \mathbf{v} + \mathbf{v}_s^e \].  

(22)

For the cases where shear flow is present at the top boundary we use periodic boundary conditions in the \( x \) direction and the following boundary conditions in the \( y \) direction.

\[ \nabla [(c \mathbf{v} \phi_s - D_s \phi_s \nabla c) \cdot \mathbf{n}]|_{y=0,1} = 0, \quad \nabla \phi_n \cdot \mathbf{n}|_{y=0,1} = 0, \]

\[ \nabla \left[ \mathbf{v} \phi_n - \Lambda \phi_n \frac{\delta f}{\delta \phi_n} \right] \cdot \mathbf{n}|_{y=0,1} = 0, \quad \mathbf{v}|_{y=0} = 0, \quad \mathbf{v}|_{y=1} = \mathbf{v}_0 \]  

(23)

3.2.4 NONDIMENSIONALIZATION

The system of equations will be nondimensionalized using a characteristic time-scale, \( t_0 \), and length-scale, \( h \). The values are specified in a table below in the results section. The nondimensionalized variables are as follows:

\[ \tilde{t} = \frac{t}{t_0}, \quad \tilde{x} = \frac{x}{h}, \quad \tilde{\mathbf{v}} = \frac{\mathbf{v} t_0}{h}, \quad \tilde{\rho} = \frac{\rho t_0^2}{\rho_0 h^2}, \quad \tilde{\tau}_n = \frac{\tau_n t_0^2}{\rho_0 h^2}, \quad \tilde{c} = \frac{c}{c_0} \]  

(24)
with $c_0$ being a characteristic substrate concentration. Using these variables the following nondimensional parameters are found [54]

$$\Lambda = \frac{\lambda \rho_0}{t_0}, \quad \Gamma_1 = \frac{\gamma_1 k T t_0^2}{\rho_0 h^4}, \quad \Gamma_2 = \frac{\gamma_2 k T t_0^2}{\rho_0 h^4},$$

$$Re_s = \frac{\rho_0 h^2}{\eta_s t_0}, \quad Re_n = \frac{\rho_0 h^2}{\eta_n t_0}, \quad \bar{D}_s = \frac{D_s t_0}{h^2}, \quad \bar{D}_n = \frac{D_n t_0}{h^2},$$

(25)

where $\rho_0$ is the averaged density; $Re_{s,n,p}$ are the Reynolds numbers for the solvent and the polymer network. The $\Lambda$, $\Gamma_{1,2}$, $\bar{D}_s$, and $\bar{\mu}$ are the dimensionless versions of the same named dimensional parameters. The Deborah number, $\Lambda_1$, is a number that is used to characterize the fluidity of materials [43]. In this case it is used in the dimensionless version of the differential equation to solve for $\tau_n$ from equation (18).

For simplicity, the $\sim$ is dropped and the dimensionless equations are then

$$\nabla \cdot \mathbf{v} = 0,$$

$$\rho \frac{d\mathbf{v}}{dt} = \nabla \cdot (\phi_n \tau_n + \phi_s \tau_s) - [\nabla p + \Gamma_1 (\nabla \phi_n \nabla \phi_n)],$$

$$\frac{\partial}{\partial t} (\phi_s c) + \nabla \cdot (c \mathbf{v} \phi_s - D_s \phi_s \nabla c) = -g_c,$$

$$\frac{\partial \phi_n}{\partial t} + \nabla \cdot (\phi_n \mathbf{v}) = \nabla \cdot \left( \Lambda \phi_n \nabla \frac{\delta f}{\delta \phi} \right) + g_n,$$

$$f(\phi_n) = \frac{\Gamma_1}{2} ||\nabla \phi_n||^2 + \Gamma_2 \left( \phi_n^2 (1 - \phi_n)^2 \right)$$

(26)

### 3.3 NUMERICAL METHODS

Looking to equation (26) as a starting point, the continuity and momentum transport equations are discussed. The momentum transport and continuity equations are solved using a velocity corrected projection method. The projection method first proposed by
Alexandre Chorin in 1967 works equally well for both 3D and 2D [6]. This method decouples the equations so they are easier to solve. It only requires solving two decoupled equations for pressure and velocity which makes it efficient for numerical simulations [19].

The projection method proposed by Chorin uses the incompressible Navier-Stokes equation as a starting point. To reformulate the momentum equation to fit the projection method, let

\[
R = -\nabla \cdot \left( \Gamma_1 \nabla \phi_n \nabla \phi_n \right) + \nabla \cdot \left( \phi_n \tau_n + \phi_s \tau_s - \frac{2}{Re_a} D \right) \tag{27}
\]

where \( D \) is as defined in equation 19 and \( Re_a \) is the volume fraction averaged Reynolds number. This then gives rise to the momentum transport equation as

\[
\rho \left( \frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} \right) = -\nabla p + \frac{1}{Re_a} \nabla^2 \mathbf{v} + R. \tag{28}
\]

Using the projection method, \( \mathbf{v} \) can be determined using the following process. If the pressure is ignored in equation (28), the following boundary value problem for \( u^{n+1} \) can be solved.

\[
\frac{u^{n+1} - v^n}{\Delta t} = -v^n \cdot \nabla v^n + \frac{1}{\rho Re_a} \nabla^2 v \tag{29}
\]

\[
\left. u^{n+1} \right|_{y=0} = 0, \quad \left. u^{n+1} \right|_{y=1} = u_0
\]

This will result in a velocity, \( u^{n+1} \), that is not divergence free. Next, the pressure is computed at the next time step to correct the velocity. The following Poisson equation with Neumann boundary conditions must be solved to determine the pressure.

\[
-\nabla \cdot \frac{1}{\rho^{n+1}} \nabla p^{n+1} = \nabla \cdot u^{n+1} \tag{30}
\]

\[
\left. \frac{\partial p^{n+1}}{\partial n} \right|_{y=0,1} = 0
\]
With this updated pressure, the velocity is updated to enforce the divergence free condition using equation 31.

\[ v^{n+1} = u^{n+1} + \frac{1}{\rho^{n+1}} \nabla p^{n+1} \]  

(31)

Central differences are used for the spatial discretization to ensure second order accuracy in space. In order to get second order accuracy in time, Crank-Nicholson and extrapolation are employed for the non-linear terms in \( R \) and \( f \). A structured grid with uniform mesh size is used in both space and time. The time step size is denoted using \( \Delta t \) and the spatial step sizes are denoted as \( \Delta x \) and \( \Delta y \). The computational domain \( \Omega = [0,1] \times [0,1] \) is divided using nodes located at \( (x_i, y_j) = (i\Delta x, j\Delta y) \) with \( i = 0, 1, \ldots, N_x \) and \( j = 0, 1, \ldots, N_y \). At the node points \( (n\Delta t, i\Delta x, j\Delta y) \), the solutions are denoted using a superscript for time and subscripts for space. Denote the network volume fraction as \( \phi^n_{i,j} \) and for the nutrient concentration use \( c^n_{i,j} \). For several of the cases covered in the results section 3.5, the boundary condition \( v \cdot n \mid_{y=0,1} = 0 \) is employed. This results in the boundary conditions for \( \phi \) and \( c \) in equation (23) becoming

\[ \nabla c \cdot n \mid_{y=0,1} = 0, \quad \nabla \phi \cdot n \mid_{y=0,1} = 0, \quad \nabla \frac{\delta f}{\delta \phi} \cdot n \mid_{y=0,1} = 0. \]  

(32)

This would represent a zero flux of these quantities through the corresponding surface. These boundary conditions result in the discrete boundary conditions given by the following equations.

\[ \phi^n_{i,1} = \phi^n_{i,-1}, \quad \phi^n_{i,2} = \phi^n_{i,-2}, \]

\[ \phi^n_{i,N_y+1} = \phi^n_{i,N_y-1}, \quad \phi^n_{i,N_y+2} = \phi^n_{i,N_y-2}, \]  

(33)

\[ c^n_{i,1} = c^n_{i,-1}, \quad c^n_{i,N_y+1} = c^n_{i,N_y-1} \]
3.4 PARALLEL IMPLEMENTATION

Discretization of system (26) results in a sequence of linear algebraic systems. For high fidelity simulations, numerical solution of these systems is intractable without the use of parallel computing. As in the previous section, PETSc and MPI are used to parallelize the data structures and iteratively solve the linear systems. However, special care is required for the all-Neumann boundary condition case for the Poisson equation 30, which results in a singular system. This occurs, for example, when velocities are prescribed at all boundaries of the domain. To remedy this, PETSc allows the removal the null space of constant functions, which makes the problem nonsingular.

Listing 5: Removing a nullspace in PETSc.

```c
MatNullSpaceCreate (PETSC_COMM_WORLD, PETSC_TRUE, 0, 0, &nsp);
MatSetNullSpace (A, nsp);
MatNullSpaceRemove (nsp, b);
```

The parallel performance of the implementation is evaluated with a strong scaling test. The scaling and speedup relative to the single processor case are shown below for a test problem on a 256x256 and 512x512 grid. For the 256x256 grid, good scaling is seen up to 3 nodes, when parallel communication costs begin to degrade the performance. On the other hand, the 512x512 case has a better computation to communication ratio, and as such it scales nearly ideally out to 8 nodes.

3.5 SIMULATION RESULTS

The time evolution of the biofilm is simulated in a variety of settings. For the result presented here we used periodic boundary conditions in the $x$ direction to be able to view downstream effects. The dimensionless parameter values shown below are used for all
Figure 15: Strong scaling raw simulation times compared to an ideal scaling reference line.

Figure 16: Strong scaling speedup compared to the ideal case.

Re = 9.98 × 10^{-4}, \quad Re_n = 2.33 \times 10^{-9}, \quad \Lambda = 10^{-10},
\Gamma_1 = 33.467, \quad \Gamma_2 = 1.25 \times 10^6, \quad D_s = 2.3.
\mu = 0.14, \quad K_c = 0.15, \quad A = 100

A table of all the dimensional parameters is included below.
Table 4: List of parameters values used for the simulations.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T$</td>
<td>Temperature</td>
<td>303</td>
<td>K</td>
</tr>
<tr>
<td>$\gamma_1$</td>
<td>Distortional energy</td>
<td>$5 \times 10^7$</td>
<td>kgm$^{-1}$s$^{-2}$</td>
</tr>
<tr>
<td>$\gamma_2$</td>
<td>Separation energy</td>
<td>$1 \times 10^{16}$</td>
<td>kgm$^{-1}$s$^{-2}$</td>
</tr>
<tr>
<td>$\chi$</td>
<td>Flory-Huggins parameter</td>
<td>0.58</td>
<td></td>
</tr>
<tr>
<td>$\lambda_{CH}$</td>
<td>Mobility parameter</td>
<td>$1 \times 10^{-10}$</td>
<td>kg$^{-1}$m$^3$s</td>
</tr>
<tr>
<td>$N$</td>
<td>Generalized polymerization parameter</td>
<td>$1 \times 10^3$</td>
<td></td>
</tr>
<tr>
<td>$\mu$</td>
<td>Max production rate</td>
<td>$1.4 \times 10^{-4}$</td>
<td>kgm$^{-3}$s$^{-1}$</td>
</tr>
<tr>
<td>$A$</td>
<td>Max consumption rate</td>
<td>0.1</td>
<td>kgm$^{-3}$s$^{-1}$</td>
</tr>
<tr>
<td>$D$</td>
<td>Diffusion coefficient.</td>
<td>$2.3 \times 10^{-9}$</td>
<td>m$^2$s$^{-1}$</td>
</tr>
<tr>
<td>$\eta_n$</td>
<td>Dynamic viscosity of the network</td>
<td>$4.3 \times 10^2$</td>
<td>kgm$^{-1}$s$^{-1}$</td>
</tr>
<tr>
<td>$\eta_s$</td>
<td>Dynamic viscosity of the solvent</td>
<td>$1.002 \times 10^{-3}$</td>
<td>kgm$^{-1}$s$^{-1}$</td>
</tr>
<tr>
<td>$\rho_n$</td>
<td>Network density</td>
<td>$1 \times 10^3$</td>
<td>kg$^{-1}$</td>
</tr>
<tr>
<td>$\rho_s$</td>
<td>Solvent density</td>
<td>$1 \times 10^3$</td>
<td>kg$^{-1}$</td>
</tr>
<tr>
<td>$t_0$</td>
<td>Characteristic time scale.</td>
<td>$1 \times 10^3$</td>
<td>s</td>
</tr>
<tr>
<td>$h_0$</td>
<td>Characteristic length scale</td>
<td>$1 \times 10^{-3}$</td>
<td>m</td>
</tr>
<tr>
<td>$c_0$</td>
<td>Characteristic substrate concentration.</td>
<td>$1 \times 10^{-3}$</td>
<td>kgm$^{-3}$</td>
</tr>
<tr>
<td>$M$</td>
<td>number of spatial intervals</td>
<td>256</td>
<td></td>
</tr>
</tbody>
</table>

3.5.1 2D FLUID FLOW WITH BIOFILM PROTRUSIONS

First, a case is considered to test the parallel code and compare it against previous results in the serial code. In this case a pair of mushroom shaped protrusions of biofilm extend away from the base. There is a lower concentration of biofilm in the neck region of the protrusion when compared to the top. For this simulation, periodic boundary conditions are prescribed at the left and right boundaries and no flux of the polymer network at the top and bottom. A shear velocity is imposed at the top of the domain

$$u\big|_{y=1} = 0.1 \quad v\big|_{y=1} = 0. \quad (35)$$

As can be seen in figure 17, the mushroom shaped regions are first deformed by the fluid flow. Both regions undergo a stretching as well as attenuation. This is easier to see by comparing the large circular region at the top of the first protrusion across different
timesteps. Eventually the mushroom shaped regions break away from the base. While this simulation is different from previous work it does exhibit the expected end behavior of shedding a region of biofilm that is connected through a thin neck to the bulk biofilm as described in [53].

![Image](image_url)

Figure 17: The biofilm is now allowed to change in both the horizontal and vertical directions. In part (a) we see the initial condition. In part (b) we see the result after 20 timesteps or the equivalent of 5.5 hours. Part (c) the initial condition has been stretched out and is starting to thin near the attachment point after 200 timesteps. Part (d), after timestep 500 equivalent to 5.8 days, the biofilm has broken away from the base layer of material and is now moving freely in the flow.

### 3.5.2 BIOFILM DROPLET VALIDATION CASE

The equation used for this work is the *modified* Cahn-Hilliard equation, which has different numerical properties than the Cahn-Hilliard equation. Zhang et. al. have shown that the Cahn-Hilliard equation produces strong dissipation for the simple case of two viscous droplets immersed in a less viscous medium [55]. An analogous situation is two
biofilm droplets immersed in water. This situation is shown below in figure 18. While there is more dissipation than reported by Zhang [55], the shape and size of the droplets is largely preserved. This difference could be due to differing time integration schemes.

Figure 18: A 256x128 simulation of two viscous biofilm droplets immersed in a less viscous medium.

Preliminary results for the analogous 3D case have also been collected and shown in figure 19. Instead of biofilm circles immersed in a 2D medium, two biofilm spheres are suspended in water. Greatly enhanced dissipation is observed, with the smaller sphere disappearing completely. The cause of this is probably that the nondimensionalization must be modified by introducing several factors of the mesh size into the denominators of equations (25) to match the model parameters in the 2D case. From investigating a multitude of numerical cases, it can be seen that the magnitude of the network Reynolds
number has a strong affect on the dissipation. Thus, if there is a mismatch between the Reynolds number in the 2D and 3D cases, differing levels of dissipation should be expected. Despite this incongruity, the overall behavior of the simulation is as expected.

Figure 19: A 256x256x128 simulation of two viscous biofilm droplets suspended in a less viscous medium.

Figure 20: A 256x256x128 simulation of two biofilm droplets, $t = 2$, with slices to visualize the inner concentration.
3.5.3 3D FLOW AROUND A BIOFILM

Several target simulations are being investigated for the 3D code, although the simulation and analysis takes a significant amount of time due to the huge amount of data. One example is as follows. With the goal of mimicking a more natural biofilm formation, preliminary results of a mottled biofilm surface of varying concentration are shown below. The white streamlines show the nontrivial flow patterns that emerge as a result of the biofilm-fluid coupling.

![Image: Streamlines over a bumpy biofilm surface.](image)

Figure 21: Streamlines over a bumpy biofilm surface.

3.6 BIOFILMS FUTURE WORK

A parallel implementation of the finite difference method for the modified Cahn-Hilliard equation has been presented. The parallel implementation scales nearly ideally out to 8 nodes, and further scaling is expected for larger meshes. The simulations exhibit the expected behavior long term. The protrusion of biofilm is connected by a thin neck to the bulk biofilm and eventually breaks away. The pieces that have broken off then merge together into a single mass of biofilm and continue to move with the fluid flow present. Work is currently underway on the combined Navier-Stokes-Cahn-Hilliard 3D model that will allow us to simulate more realistic geometries and flow fields. We are also investigating implementing preconditioners based on domain decomposition methods [25, 26, 56] in order to design an efficient parallel simulation procedure.
4 SIMULATION OF THE ALTERNATING CURRENT OPTIMAL POWER FLOW PROBLEM

The penetration of distributed energy resources, especially from renewable sources, is rising globally. For example, in the U.S., renewable energy sources (REs) provided 17% of total electricity generation in 2017 [12]. Renewable portfolio standards require independent power producers to have certain percentages of electricity generation from REs. For example, it is required to have 30% electricity generation from REs in Hawaii and 33% in California by 2020 [32]. The stochastic nature of the generation profile from most of REs pose a challenge to maintaining a reliable, secure, and economically efficient power system operation. In light of these circumstances, currently a major problem is to find the optimal operating state of all generating sources under varying load and generation conditions to maintain reliable, secure, and efficient power system operation.

The problem of finding optimal dispatch of generators that satisfies the underlying network and other constraints is termed AC optimal power flow (ACOPF). The Federal Energy Regulatory Commission (FERC) estimates that a 5% increase in ACOPF solution accuracy from a 2009 base case can result in up to $20 billion and $90 billion savings per year in the US and worldwide, respectively. ACOPF minimizes a cost function with respect to the vector of continuous optimization variables, such as nodal voltages, generator active power, generator reactive power, etc. — the details of which can be found in [49]. The solution to ACOPF also satisfies constraints such as line limits, generation (both active and reactive power) limits, and voltage limits, among other quantities.

HPC is gaining popularity in power system applications, such as optimization, real-time control and simulation, and integrated transmission-distribution simulations [13, 18, 35]. In an approach investigated by [24, 3], a given power system network is decomposed into multiple overlapping regions, each having regional optimal power flow (OPF) problems that can be solved by multiple cores. This would limit the number of
overlapping regions depending upon the nature of constraints, especially in case of constraints that involve the core variables from both overlapping regions and could limit the number of cores that can be used. Tools such as DMNetwork (implemented in PETCs) and PLASMO (Platform for Scalable Modeling and Optimization) that use graph based abstraction for model representation and interaction between submodels are also becoming popular for solving OPF problems [23]. In this paper, a scalable parallel implementations for the ACOPF optimization problem is explored using the Portable Extensible Toolkit for Scientific Computation (PETSc), especially by leveraging PETSC’s DMNetwork library focusing on large-scale power system networks.

The primal-dual interior point method (PDIPM) is used to solve ACOPF. A major advantage of the PDIPM approach [22] compared to other algorithms is its insensitivity to problem size regarding the number of iterations required to reach to the optimal solution. Solving non-linear optimization problems using PDIPM essentially requires solving a linear system of equations, which is very suitable for parallel implementation in multiple cores using HPC.

The remaining topics of this section are organized as follows. Section 4.1 describes the formulation of ACOPF as a non-linear optimization problem. In Section 4.2, the PDIPM utilized in solving non-linear optimal power flow problems is explained. Details about the parallel implementation of ACOPF using PETSc and DMNetwork are provided in Section 4.3. Section 4.4 discusses the different power system test cases that the parallel implementations are tested on, including optimization parameters. Section 4.5 presents the simulation results and analysis on the parallel performance (e.g., execution time, speedup, and solution accuracy) of the developed algorithms for solving the ACOPF problem. Conclusions and potential future work is discussed in Section 4.6.
4.1 FORMULATION OF ACOPF

Consider a transmission network with \( N_B \) buses collected in set \( B := \{1, 2, \ldots, N_B\} \) and \( N_L \) transmission lines collected in set \( L \subset B \times B \). Let \( Y \in \mathbb{C}^{N_B \times N_B} \) be the admittance matrix of the transmission network. Note, \( Y_{ij} = 0, ij \in B \), if there is no transmission line connecting node \( i \) and \( j \), and \( Y_{ij} = Y_{ji} \). In a real power system network, the matrix \( Y \) is sparse because \( N_L << N_B \times N_B \).

Let \( v \) and \( s \) be the column matrices of complex voltage and net injected complex power at all buses. The relation between \( v \) and \( s \) is given by (36).

\[
s = p + jq = v(Yv)^* \tag{36}
\]

where \( p \) and \( q \) represent the column vectors of real and imaginary parts of the complex power at all buses, respectively (also known as active power and reactive power). Note, the complex voltage at bus \( k \in B \), i.e., \( v_k \), can be expressed in terms of its magnitude \( u_k \) and angle \( \theta_k \) by (37).

\[
v_k = u_k \angle \theta_k \tag{37}
\]

The column vectors of voltage magnitudes and angles will be denoted as \( u \) and \( \theta \in \mathbb{R}^{N_B \times 1} \), respectively. The voltage angle at any bus is with respect to the angle of the single reference bus (typically index 1), and its voltage angle will be zero. Furthermore, active power generation sources are indexed by the set \( P := \{1, 2, \ldots, N_P\} \), and reactive power generation sources indexed by the set \( Q := \{1, 2, \ldots, N_Q\} \). Now an active power injection matrix, \( C_P \in \mathbb{R}^{N_B \times N_P} \), is defined in such a way that

\[
(C_P)_{ki} = 1, \forall k \in B, \forall i \in P, \text{ if active power generation source } i \text{ is connected to bus } k,
\]

else 0. The reactive power injection matrix \( C_Q \) can be similarly defined. We also define \( p_G, p_D, q_G, \) and \( q_D \in \mathbb{R}^{N_P \times 1} \) as column vectors of active power generation, active power

\[1\]Upper-case (lower-case) boldface letters will be used for matrices (column vectors); \((\cdot)^T\) for matrix transposition; \(\text{Re}(\cdot)\) and \(\text{Im}(\cdot)\) denote the real and imaginary parts of a complex number, respectively; \(j := \sqrt{-1}\) the imaginary unit; for a given column vector \( x \), \( x_i \) represents \( i^{th} \) entry.
demand, reactive power generation, and reactive power demand, respectively.

The AC power flow equation (one of the network constraints that needs to be satisfied) can then be written as

\[ C_p p_G - p_D - \text{Re}(v(Yv)^*) = 0, \]  
\[ (38) \]

\[ C_Q q_G - q_D - \text{Im}(v(Yv)^*) = 0. \]  
\[ (39) \]

The transmission line is limited by its thermal capacity to carry power. Let \( S_{ij}, \forall i \neq j \in \mathcal{B}, Y_{ij} \neq 0 \) be the upper limit of power flow over a transmission line connecting buses \( i \) and \( j \). To formulate this as one of the constraints, we have to consider the power flow through the line from both directions (i.e., power flowing from \( i \) to \( j \), and from \( j \) to \( i \)). If \( s_i \) and \( s_j \) are the complex power injected into line \( ij \) (if this line exists) from buses \( i \) and \( j \), respectively, then (40) and (41) need to be satisfied.

\[ s_i s_i^* - S_{ij}^2 \leq 0, \forall i \neq j \in \mathcal{B}, Y_{ij} \neq 0 \]  
\[ (40) \]

\[ s_j s_j^* - S_{ij}^2 \leq 0, \forall i \neq j \in \mathcal{B}, Y_{ij} \neq 0 \]  
\[ (41) \]

Other constraints to be satisfied are zero angle for reference bus, the active and reactive power sources also have upper and lower bounds on their generation capacity, and the voltage magnitude should not violate overvoltage and undervoltage thresholds.

\[ \min_{\theta, u, p_G, q_G} f(p_G) \]  
\[ (42) \]

s.t., (38), (39), (40), (41)

\[ \theta_1 = 0 \]  
\[ (43) \]

\[ u_{\text{min}} \leq u_k \leq u_{\text{max}}, \forall k \in \mathcal{B} \]  
\[ (44) \]
\[ p_{G,k,\text{min}} \leq p_{G,k} \leq p_{G,k,\text{max}}, \forall k \in \mathcal{P} \]  
(45)

\[ q_{G,k,\text{min}} \leq q_{G,k} \leq q_{G,k,\text{max}}, \forall k \in \mathcal{Q} \]  
(46)

where \( u_{\text{min}}, u_{\text{max}} \) represent lower and upper bound on voltage magnitudes, and lower and upper bounds on active power and reactive power at bus \( k \) are denoted by \( p_{G,k,\text{min}}, p_{G,k,\text{max}}, q_{G,k,\text{min}} \) and \( q_{G,k,\text{max}} \), respectively.

ACOPF is thus a non-linear optimization problem that seeks to minimize the cost of generation in a given power system, defined by (42), subject to constraints (38) – (41) and (43) – (46). Note that in this standard formulation, the cost function is only a function of active (real) power output of the generators, and typical cost functions of generating units are quadratic.

### 4.2 PRIMAL DUAL INTERIOR POINT METHOD

This section gives a brief overview of the PDIPM method used to solve the ACOPF optimization problems. Consider a non-linear optimization problem with cost function \( f(x) \), equality constraints \( g(x) \), and inequality constraints \( h(x) \). In general, optimization problems (e.g., ACOPF) can be rewritten in a more compact form, as given in (47).

\[
\begin{align*}
\min_x & \quad f(x) \\
\text{s.t.} & \quad g(x) = 0 \\
& \quad h(x) \leq 0
\end{align*}
\]  
(47)

The optimization problem in (47) is converted into an equivalent single Lagrangian function in [57, 16], and shown in (48). The variables \( \lambda, \mu, \) and \( z \) are vectors collecting Lagrangian multipliers for equality constraints, Lagrangian multipliers for inequality constraints, and slack variables, respectively. Let \( \gamma \) be the perturbation
parameter in the barrier function. Here, \( n_h \) is the number of inequality constraints.

\[
\mathcal{L}^\gamma = f(x) + \lambda^T g(x) + \mu^T (h(x) + z) - \gamma \sum_{i=1}^{n_h} \ln(z_i)
\]  

(48)

Solving (48) requires satisfying Karush-Kuhn-Tucker (KKT) or first-order optimality conditions, which further requires setting all first-order partial derivatives of the Lagrangian function to zero\(^2\). Note, \( \mu \) and \( z \) should be positive. The iterative method requires solving (49) using a reduced KKT matrix [57].

\[
\begin{pmatrix}
\mathcal{L}_{xx} + h_x^T [z]^{-1} [\mu] h_x & g_x^T \\
g_x & 0
\end{pmatrix}
\begin{pmatrix}
\Delta x \\
\Delta \lambda
\end{pmatrix}
= 
\begin{pmatrix}
\Delta x \\
\Delta \lambda
\end{pmatrix}
\]  

(49)

Here, \( \Delta x \) and \( \Delta \lambda \) are the changes in the primal and dual variables, respectively. \( \Delta x \) indicates the changes in the primal variables, and \( \Delta \lambda \) indicates the changes in the dual variables. \( \mathcal{L}_{xx} \) is the Hessian matrix of the Lagrangian function, \( h_x^T [z]^{-1} [\mu] h_x \) indicates the second-order partial derivative of the Lagrangian function, \( g_x^T \) indicates the gradient of the augmented function, and \( g_x \) indicates the gradient of the constraints function.

The next sections describe how to improve the performance of existing preconditioners in solving such linear systems of equations involving ill-conditioned matrices. The goal is to improve both the solution time and solution accuracy of the parallel implementation of ACOPF using PDIPM.

4.3 PARALLEL IMPLEMENTATION

PETSc is used to parallelize the power flow data structures and computations. Specifically, PETSc’s DMNetwork data management object for distributed networks is used to represent the power grid of buses, connections, loads, resistances, and other pertinent features. The DMNetwork object provides an interface to algebraic solvers within PETSc, greatly simplifying the distribution of network data over many MPI processes [30].

\(^2\)If \( f \) is a general multivariate continuous function, then \( f_x \) indicates the first-order partial derivative of \( f \) with respect to variable \( x \), and \( f_{xx} \) indicates the second order partial derivative of function \( f \) with respect to \( x \).
The power grid network data is initially read in on one MPI process. User-defined data structures are registered with a DMNetwork object using PETSc’s function DMNetworkRegisterComponent, as shown in Listing 6 below\(^3\). For example, we define a bus\_t data structure which describes the active and reactive loads on the power grid.

Listing 6: Register user-defined data structures with the DMNetwork object.

\[
\text{DMNetworkRegisterComponent(DMNetwork, "bus\_data", sizeof(bus\_t), &bus\_key);}
\]

\[
\text{DMNetworkRegisterComponent(DMNetwork, "edge\_data", sizeof(edge\_t), &edge\_key);}
\]

Memory is allocated for user-defined data structures at particular nodes or edges in the network using DMNetworkAddComponent. The flexibility to add a variable number of “components” to each vertex is useful because a vertex in the network may be associated with both a bus and a generator. Initially, the network data resides on the root process, but in general the network is decomposed among the processors. Each process owns a subset of the network which represents the power grid, and the extent of this subset can be

---

\(^3\)Listing code is at times abridged to improve readability and concision.
queried with `DMNetworkGetEdgeRange` and `DMNetworkGetVertexRange`. Listing 7 shows memory being allocated for the bus data at the network vertices.

Listing 7: Allocate memory for the bus data structure at network vertices.

```c
PetscInt vStart, vEnd;
DMNetworkGetVertexRange(DMNetwork,& vStart,& vEnd);
for (int i = vStart; i < vEnd; i++){
    DMNetworkAddComponent(DMNetwork, i, bus_key, bus_array[i-vStart]);
}
```

Above, `bus_key` is an integer that allows correct memory addressing in the `DMNetwork` object, and `bus_array` is an array of structs that holds the bus data. Similarly, memory is allocated for the network edge data.

Listing 8: Allocate memory for the edge data structure at network edges.

```c
PetscInt eStart, eEnd;
DMNetworkGetEdgeRange(DMNetwork,& eStart,& eEnd);
for (int i = eStart; i < eEnd; i++){
    DMNetworkAddComponent(DMNetwork, i, edge_key, edge_array[i-eStart]);
}
```

After registering the generators, buses, loads, and branches, the network is distributed using `DMNetworkDistribute`, which passes a subset of the network to each process. Using the distributed network, the matrices and vectors needed in the primal dual interior point method are constructed in parallel. Pseudo-code is given in Listing 9 to illustrate creating a matrix using the network data. A more detailed example can be found in the `admMat.c` source file of the repository listed in Section 4.

Listing 9: Constructing a matrix using network data.

```c
Mat A;
PetscInt eStart, eEnd;
```
edge_t edge;
DMNetworkGetEdgeRange(net,&eStart,&eEnd);
for (int i = eStart; i < eEnd; i++){
    DMNetworkGetComponent(DMNetwork,i,&edge);
    MatSetValue(A, edge->edge_id, edge->impedence, 1, INSERT_VALUES);
}
MatAssemblyBegin(A, MAT_FINAL_ASSEMBLY);
MatAssemblyEnd(A, MAT_FINAL_ASSEMBLY);

It should be noted that the parallel decomposition of the network and any given matrix or vector are not necessarily conformal. If Processor 0 owns the first vertex in the network, there is no guarantee that it also owns the first row of some matrix A. This means that during the PETSc MatAssembly phase, communication may be required to assign matrix elements to the processors that own the corresponding matrix rows.

The current primal dual interior point implementation uses Krylov subspace solvers (KSP) and preconditioners (PCs) to solve the linear system described in (49). In simplified notation, problem (49) is of the form \(Ax = b\). PETSc provides the KSP and PC objects for solving such linear systems, as shown in Listing 10.

Listing 10: Solving the linear system using PETSc KSP, PC objects.

Mat A;
Vec x, b;
KSP ksp;
PC pc;
KSPCreate(PETSC_COMM_WORLD,&ksp);
KSPSetOperators(ksp,A,A);
KSPGetPC(ksp,&pc);
PCSetType(pc,PCASM);
PCSetUp (pc);
KSPSetUp (ksp);
// ...
// Construct matrix A and vector b
// ...
KSPSolve (ksp, b, x);

4.4 SIMULATION PARAMETERS

Three IEEE standard power system networks, namely 118 bus, 300 bus, and 1354 bus systems, are used to evaluate the parallel implementation of ACOPF. The generator information, line parameters, and load data for these systems were collected from MATPOWER, and the solution accuracy is compared with the result from the runopf function within MATPOWER [58]. The parallel algorithm is implemented in PETSc and run on Blackjack cluster of South Dakota State University (SDSU). This particular cluster has 50 nodes, with each node having 12 cores/threads composed of two 2.93 GHz Intel Xeon X5670 processors (6 cores each) with 48 GB of RAM. The processors use a non-uniform memory access (NUMA) architecture. The cluster uses Mellanox InfiniBand low-latency interconnect for inter-node communication.

4.5 PARALLEL PERFORMANCE

The execution time with increasing number of CPUs for three test cases are plotted in Fig. 23. To mitigate the chance of bad runs, the simulation was executed three times with the average times for execution plotted for each case. Fig. 23 shows that the average execution time reduces for 300 bus and 1354 bus test case systems, however for the 118 bus case it reduces until 8 cores. One reason this occurs could be that the problem size for 118 bus is too small so that the communication time overhead is much larger than the computation time, making it unsuitable after a certain number of CPUs for parallelism.
The reduced average execution time for higher test case systems is promising in that a lot of time can be saved by using HPC to solve ACOPF problems. Strong scaling results for the parallel ACOPF implementation with an Additive Schwarz (ASM) preconditioner are shown in Fig. 24. For networks large enough to provide a high computational intensity, e.g., the 1354 bus and 300 bus systems, the implementation shows significant speedup. However, variability is seen in performance, especially with the 300 bus system. Several factors can influence the number of iterations needed for convergence, but the primary factor in this case is the number of subdomains for the ASM preconditioner. The number of subdomains is set to the number of MPI processes, so increasing the number of CPUs can change the numerical convergence rate. The dependence on preconditioner is further illustrated by comparing against the performance of using PETSc’s Jacobi preconditioner or no preconditioner at all – in both cases, the linear solver failed to converge within several hundred thousand iterations. In other words, the problem is extremely numerically stiff without an effective preconditioner. Given the apparent importance of the preconditioner for algorithm performance, a brief discussion of preconditioning and several numerical studies are discussed now. The PDIPM problem in (49) is indefinite and
generally ill-conditioned [16]. This linear system can be written in general form as

$$Ax = b \text{ where } A = \begin{pmatrix} H & B^T \\ B & 0 \end{pmatrix}, \quad H \in \mathbb{R}^{n \times n}, \quad B \in \mathbb{R}^{n^\lambda \times n^\sigma}, \quad n^\lambda \leq n^\sigma. \quad (50)$$

To solve (50) for large problems, parallel iterative methods such as the Conjugate Gradient (CG) method [21] or the Generalized minimal residual (GMRES) method [45]. Both CG and GMRES are examples of Krylov Subspace Projection (KSP) methods, a class of iterative linear solvers. The convergence rates of these methods are inversely related to the condition number of the matrix. Thus, for an ill-conditioned problem, the convergence rate will suffer. Preconditioning can mitigate the condition number of an ill-conditioned problem. In rough terms, preconditioning means replacing the system $Ax = b$ with a system that is more easily solved. One might replace $Ax = b$ with $M^{-1}Ax = M^{-1}b$ where $M$ is a matrix such that the condition number of $M^{-1}A$ is smaller than the original matrix. As an extreme case, $M^{-1} = A^{-1}$ would be an excellent conditioner. More commonly, $M = \text{diag}(A)$ is the Jacobi preconditioner. For the PETSC ASM preconditioner used in this study, we investigate the affect of overlap size on the PDIPM algorithm convergence. In figure 25 the residual for the PDIPM algorithm and number of GMRES iterations per outer loop is reported. The number of processors
indicates the number of subdomains for the ASM preconditioner. In addition, the affect of

overlap on overall runtime for the 1300 bus test system is reported in figure 26.
4.6 ACOPF FUTURE WORK

Different methods of decomposing power system networks for effective parallel implementation might need more detail understanding of problem itself and also in-depth knowledge of new tools such as PETSc and PLASMO. Additionally, synthetic generator information and load data which closely resemble the real load and generator data are necessary [20, 11]. Finally, the different approaches will be tested to achieve the best scalable parallel performance for ACOPF. Given the overall dependence of the computation time and scalability on the preconditioner for the linear system, we will investigate the performance benefits of user-defined preconditioners based on domain decomposition methods that better consider the underlying characteristics of the network.

Figure 26: Affect of ASM overlap on ACOPF runtime.
5 CONCLUSION

As the importance of computing to the scientific method continues to grow, broader understanding is needed of the methods and challenges of designing parallel scientific codes. To that end, several applications in high performance computing have been presented. The analysis of impinging jets using DNS provides the potential to better understand the physics of this fundamental flow pattern and, with an enhanced understanding, design better thermal and mass transfer systems. Modeling and simulation of biofilms is a challenging mathematical problem, and the numerical investigation of 2D and 3D cases allows greater understanding of both the model itself and the numerical properties of the model. Finally, numerically solving the ACOPF nonlinear optimization problem gives a perspective on the implementation and numerical concerns presented by a situation that is modeled by a discrete network rather than partial differential equations. In each of the above cases, parallel computing was central and, indeed, necessary to obtain results of reasonable fidelity in a reasonable time frame.
REFERENCES


https://books.google.com/books?id=LP8oAQAAMAAJ.


APPENDIX

A BIOFILM DISCRETIZATION

Recall again the momentum equation (E1), the nutrient transport equation (E2), and the biofilm transport equation (E3).

\[
\rho \frac{dv}{dt} = \nabla \cdot (\phi_n \tau_n + \phi_s \tau_s) - [\nabla p + \Gamma_1 (\nabla \phi_n \nabla \phi_n)] \tag{E1}
\]

\[
\frac{\partial \phi_n}{\partial t} + \nabla \cdot (\phi_n \mathbf{v}) = \nabla \cdot \left( \Lambda \phi_n \nabla \frac{\delta f}{\delta \phi} \right) + g_n. \tag{E2}
\]

\[
f(\phi_n) = \frac{\Gamma_1}{2} ||\nabla \phi_n||^2 + \Gamma_2 (\phi_n^2 (1 - \phi_n)^2)
\]

\[
\frac{\partial}{\partial t} (\phi_s \mathbf{c}) + \nabla \cdot (\mathbf{c} \phi_s - D_s \phi_s \nabla \mathbf{c}) = -g_c \tag{E3}
\]

We will describe the discretization of these equations below. First we consider equation (E1).

A.1 EQUATION E1

\[
\rho \frac{dv}{dt} = \nabla \cdot (\phi_n \tau_n + \phi_s \tau_s) - [\nabla p + \Gamma_1 (\nabla \phi_n \nabla \phi_n)] \tag{E1}
\]

where

\[
\tau_n = \frac{2}{Re_n} \mathbf{D} = 2 \eta_n \mathbf{D},
\]

\[
\tau_s = \frac{2}{Re_s} \mathbf{D} = 2 \eta_s \mathbf{D},
\]

and

\[
\mathbf{D} = \frac{1}{2} [\nabla \mathbf{v} + \nabla \mathbf{v}^T].
\]
Following the projection method, the pressure gradient is ignored and $E_1$ is time-evolved as

$$\rho \left( \frac{\mathbf{u}^{n+1} - \mathbf{v}^n}{\Delta t} + \mathbf{v}^n \cdot \nabla \mathbf{v}^n \right) = \frac{1}{Re_a} \left( \nabla^2 \mathbf{u}^{n+1} - \nabla s^n \right) + \mathbf{R}^n$$

where

$$\mathbf{R} = -\nabla \cdot (\Gamma_1 \nabla \phi_n \nabla \phi_n) + \nabla \cdot \left( \phi_n \tau_n + \phi_s \tau_s - \frac{2}{Re_a} \mathbf{D} \right)$$

With a little work, this can be rearranged as a Helmholtz equation.

$$\frac{\rho}{\Delta t} \mathbf{u}^{n+1} - \frac{\rho}{\Delta t} \mathbf{v}^n + \rho \mathbf{v}^n \cdot \nabla \mathbf{v}^n = \frac{1}{Re_a} \nabla^2 \mathbf{u}^{n+1} - \frac{1}{Re_a} \nabla s^n + \mathbf{R}^n$$

The equation that we will solve numerically is thus

$$\nabla^2 \mathbf{u}^{n+1} - \frac{\rho}{\eta_a \Delta t} \mathbf{u}^{n+1} = -\frac{1}{\eta_a} \left( \frac{\rho}{\Delta t} \mathbf{v}^n - \rho \mathbf{v}^n \cdot \nabla \mathbf{v}^n + \mathbf{R}^n \right) + \nabla s^n \quad (E1.1)$$

Note that all terms on the left of equation E1.1 are handled implicitly, while all terms on the right are explicit, requiring only information from the previous time-step. The most difficult part of discretizing equation E1.1 is $\mathbf{R}^n$. 

$$\mathbf{R}^n = -\nabla \cdot (\Gamma_1 \nabla \phi_n \nabla \phi_n) + \nabla \cdot \left( \phi_n \tau_n + \phi_s \tau_s - \frac{2}{Re_a} \mathbf{D} \right)$$

The superscript $n$ represents the time-step and the subscript denotes the biofilm network or substrate variables. Since everything on the right hand side is explicit at this point, we will drop the superscript. When necessary, the subscript $n$ will be dropped in place of
node indices to indicate stencil operators.

\[ -\nabla \cdot (\Gamma_1 \nabla \phi_n \nabla \phi_n) = -\frac{\delta f}{\delta \phi_n} \nabla \phi_n \]

\[ = -\frac{\delta}{\delta \phi_n} \left( \frac{\Gamma_1}{2} ||\nabla \phi_n||^2 - \Gamma_2 \left( \phi_n^2 (1 - \phi_n)^2 \right) \right) \nabla \phi_n \]

\[ = - (\Gamma_1 \Delta \phi_n - \Gamma_2 (2\phi_n - 6\phi_n^2 + 4\phi_n^3)) \nabla \phi_n \]

\[ = - (\Gamma_1 \Delta \phi_n - \Gamma_2 (2\phi_n(1 - \phi_n)(1 - 2\phi_n))) \nabla \phi_n \]

\[ = - (\Gamma_1 \Delta \phi_n - \Gamma_2 h(\phi_n)) \nabla \phi_n \]

\[ = \left( -\Gamma_1 \left( \frac{\partial^2 \phi_n}{\partial x^2} + \frac{\partial^2 \phi_n}{\partial y^2} + \frac{\partial^2 \phi_n}{\partial z^2} \right) + \Gamma_2 h(\phi_n) \right) \nabla \phi_n \]

where \( h(\phi_n) = 2\phi_n(1 - \phi_n)(1 - 2\phi_n) \). The above equation is discretized using central difference approximations for the derivatives. Now we consider the other half of \( \mathbb{R}^n \), \( R^2 \), which contains the term \( \nabla \cdot \frac{2}{Re} D^n = 2\eta \nabla \cdot D^n \). This is a tensor operation and deserves further attention. First we write out \( D \).

\[ D = \frac{1}{2} \left[ \nabla v + \nabla v^T \right] \]

\[ = \frac{1}{2} \begin{bmatrix}
\frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} & \frac{\partial u}{\partial z} \\
\frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} & \frac{\partial v}{\partial z} \\
\frac{\partial w}{\partial x} & \frac{\partial w}{\partial y} & \frac{\partial w}{\partial z}
\end{bmatrix}
+ \begin{bmatrix}
\frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} & \frac{\partial u}{\partial z} \\
\frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} & \frac{\partial v}{\partial z} \\
\frac{\partial w}{\partial x} & \frac{\partial w}{\partial y} & \frac{\partial w}{\partial z}
\end{bmatrix} \]

\[ = \frac{1}{2} \begin{bmatrix}
\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} & 2\frac{\partial v}{\partial x} + \frac{\partial v}{\partial z} & \frac{\partial w}{\partial x} + \frac{\partial w}{\partial y} \\
\frac{\partial v}{\partial x} + \frac{\partial v}{\partial y} & 2\frac{\partial v}{\partial y} + \frac{\partial v}{\partial z} & \frac{\partial w}{\partial y} + \frac{\partial w}{\partial y} \\
\frac{\partial w}{\partial x} + \frac{\partial w}{\partial y} & \frac{\partial w}{\partial z} + \frac{\partial w}{\partial z} & 2\frac{\partial w}{\partial z}
\end{bmatrix} \]
The divergence of $D$ is another tensor operation.

$$\nabla \cdot D = \left( \frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right) \cdot \frac{1}{2} \begin{bmatrix} 2 \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} & \frac{\partial u}{\partial z} + \frac{\partial w}{\partial x} \\ \frac{\partial v}{\partial x} + \frac{\partial w}{\partial y} & 2 \frac{\partial v}{\partial y} & \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} \\ \frac{\partial w}{\partial x} + \frac{\partial w}{\partial y} + \frac{\partial w}{\partial z} & 2 \frac{\partial w}{\partial z} \end{bmatrix}$$

$$= \frac{1}{2} \begin{bmatrix} 2\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial x \partial y} + \frac{\partial^2 v}{\partial z \partial x} + \frac{\partial^2 w}{\partial x \partial z} \\ \frac{\partial^2 u}{\partial y^2} + 2\frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial y \partial x} + \frac{\partial^2 w}{\partial y \partial z} \\ \frac{\partial^2 u}{\partial z^2} + \frac{\partial^2 v}{\partial z^2} + \frac{\partial^2 v}{\partial z \partial x} + \frac{\partial^2 w}{\partial z \partial y} + 2\frac{\partial^2 w}{\partial z^2} \end{bmatrix}$$

$$= \frac{1}{2} \begin{bmatrix} \Delta u + \frac{\partial}{\partial x} \left( \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} \right) \\ \Delta v + \frac{\partial}{\partial y} \left( \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} \right) \\ \Delta w + \frac{\partial}{\partial z} \left( \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} \right) \end{bmatrix}$$

$$= \frac{1}{2} \Delta \mathbf{v}$$

Therefore, we have $2\eta_a \nabla \cdot D^n = \eta_a \Delta \mathbf{v}^n$. Taking another look at $R2$, we have (dropping the superscripts denoting the time step)

$$\nabla \cdot \left( \phi_n \tau_n + \phi_s \tau_s - \frac{2}{Re_a} D \right) = \nabla \cdot \left( \phi_n 2\eta_a D + \phi_s 2\eta_a D - 2\eta_a D \right)$$

$$= 2\nabla \cdot (\mu \mathbf{D}) - \eta_a \Delta \mathbf{v}$$
where \( \mu = \phi_n \eta_n + \phi_s \eta_s \) is a weighted average viscosity. Discretizing \( \nabla \cdot (\mu \mathbf{D}) \) is difficult.

\[
\nabla \cdot (\mu \mathbf{D}) = \left( \frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right) \cdot \frac{1}{2} \mu \begin{bmatrix}
2 \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} & \frac{\partial u}{\partial z} + \frac{\partial w}{\partial x} \\
\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} & 2 \frac{\partial v}{\partial y} & \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y} \\
\frac{\partial w}{\partial x} + \frac{\partial u}{\partial z} + \frac{\partial v}{\partial z} & \frac{\partial w}{\partial y} + \frac{\partial v}{\partial z} & 2 \frac{\partial w}{\partial z}
\end{bmatrix}
\]

To illustrate the discretization of these terms, let us consider first the \( x \)-direction. Denote \( \mathbf{e}_1 = (1, 0, 0) \).

\[
2 [\nabla \cdot (\mu \mathbf{D})] \cdot \mathbf{e}_1 = 2 \frac{\partial}{\partial x} \left( \mu \frac{\partial u}{\partial x} \right) + \frac{\partial}{\partial y} \left( \mu \frac{\partial u}{\partial y} + \mu \frac{\partial v}{\partial x} \right) + \frac{\partial}{\partial z} \left( \mu \frac{\partial u}{\partial z} + \mu \frac{\partial w}{\partial x} \right)
\]

Consider the following stencil about the node \((i, j, k)\), projected onto the \( xy \) plane for ease of illustration.
We have

\[
\begin{align*}
\frac{\partial}{\partial x} \left( \frac{\partial u}{\partial x} \right) & \approx \left( \mu \frac{\partial u}{\partial x} \right)_{i+\frac{1}{2}, j} - \left( \mu \frac{\partial u}{\partial x} \right)_{i-\frac{1}{2}, j} \frac{\Delta x}{\Delta x} \\
& = \mu_{i+\frac{1}{2}, j} \frac{\Delta x}{\Delta x} - \mu_{i-\frac{1}{2}, j} \frac{\Delta x}{\Delta x} \\
& \approx \mu_{i+\frac{1}{2}, j} \left( \frac{u_{i+1,j} - u_{i,j}}{\Delta x} \right) - \mu_{i-\frac{1}{2}, j} \left( \frac{u_{i,j} - u_{i-1,j}}{\Delta x} \right) \\
& = \mu_{i+\frac{1}{2}, j} \left( \frac{u_{i+1,j} - u_{i,j}}{\Delta x} \right) - \mu_{i-\frac{1}{2}, j} \frac{u_{i,j} - u_{i-1,j}}{\Delta x^2}
\end{align*}
\]
\[
\frac{\partial}{\partial y} \left( \mu \frac{\partial v}{\partial x} \right) \approx \frac{\left( \mu \frac{\partial v}{\partial x} \right)_{i,j+\frac{1}{2}} - \left( \mu \frac{\partial v}{\partial x} \right)_{i,j-\frac{1}{2}}}{\Delta y}
\]

\[
= \frac{\mu_{i,j+\frac{1}{2}} \left( \frac{\partial v}{\partial x} \right)_{i,j+\frac{1}{2}} - \mu_{i,j-\frac{1}{2}} \left( \frac{\partial v}{\partial x} \right)_{i,j-\frac{1}{2}}}{\Delta y}
\]

\[
= \frac{\mu_{i,j+\frac{1}{2}} \frac{v_{i+\frac{1}{2},j+\frac{1}{2}} - v_{i-\frac{1}{2},j+\frac{1}{2}}}{\Delta x} - \mu_{i,j-\frac{1}{2}} \frac{v_{i+\frac{1}{2},j-\frac{1}{2}} - v_{i-\frac{1}{2},j-\frac{1}{2}}}{\Delta x}}{\Delta y}
\]

Averaging is used to find the velocities at the half-indices.

\[
v_{i+\frac{1}{2},j+\frac{1}{2}} = \frac{1}{2} \left( v_{i+\frac{1}{2},j} + v_{i+1,j+\frac{1}{2}} \right) = \frac{1}{4} \left( v_{i,j} + v_{i,j+1} + v_{i+1,j} + v_{i+1,j+1} \right)
\]

\[
v_{i-\frac{1}{2},j+\frac{1}{2}} = \frac{1}{2} \left( v_{i-\frac{1}{2},j} + v_{i,j+\frac{1}{2}} \right) = \frac{1}{4} \left( v_{i-1,j} + v_{i-1,j+1} + v_{i,j} + v_{i,j+1} \right)
\]

\[
v_{i+\frac{1}{2},j+\frac{1}{2}} - v_{i-\frac{1}{2},j+\frac{1}{2}} = \frac{1}{4} \left( v_{i+1,j} + v_{i+1,j+1} - v_{i-1,j} - v_{i-1,j+1} \right)
\]

Following this pattern, we also have

\[
v_{i+\frac{1}{2},j-\frac{1}{2}} - v_{i-\frac{1}{2},j-\frac{1}{2}} = \frac{1}{4} \left( v_{i+1,j} + v_{i+1,j-1} - v_{i-1,j} - v_{i-1,j-1} \right)
\]

and thus

\[
\frac{\partial}{\partial y} \left( \mu \frac{\partial v}{\partial x} \right) \approx \left[ \mu_{j+\frac{1}{2}} \left( \frac{1}{4} \left( v_{i+1,j} + v_{i+1,j+1} - v_{i-1,j} - v_{i-1,j+1} \right) \right) \right. \]

\[
- \mu_{j-\frac{1}{2}} \left( \frac{1}{4} \left( v_{i+1,j} + v_{i+1,j-1} - v_{i-1,j} - v_{i-1,j-1} \right) \right) \right] / (\Delta x \Delta y)
\]

Similarly, it can be shown that

\[
\frac{\partial}{\partial z} \left( \mu \frac{\partial w}{\partial x} \right) \approx \left[ \mu_{k+\frac{1}{2}} \left( \frac{1}{4} \left( w_{i+1,k} + w_{i+1,k+1} - w_{i-1,k} - w_{i-1,k+1} \right) \right) \right. \]

\[
- \mu_{k-\frac{1}{2}} \left( \frac{1}{4} \left( w_{i+1,k} + w_{i+1,k-1} - w_{i-1,k} - w_{i-1,k-1} \right) \right) \right] / (\Delta x \Delta z)
\]
Combining the above discretizations, we have

\[
2 [\nabla \cdot (\mu \mathbf{D})] \cdot \mathbf{e}_1 = 2 \frac{\partial}{\partial x} \left( \mu \frac{\partial u}{\partial x} + \mu \frac{\partial u}{\partial y} \right) + \frac{\partial}{\partial y} \left( \mu \frac{\partial u}{\partial y} + \mu \frac{\partial w}{\partial x} \right) + \frac{\partial}{\partial z} \left( \mu \frac{\partial u}{\partial z} + \mu \frac{\partial w}{\partial x} \right)
\]

\[
\approx 2 \left( \mu_{i+\frac{1}{2}} (u_{i+1} - u_i) - \mu_{i-\frac{1}{2}} (u_i - u_{i-1}) \right) / (\Delta x^2)
\]

\[
+ \left( \mu_{j+\frac{1}{2}} (u_{j+1} - u_j) - \mu_{j-\frac{1}{2}} (u_j - u_{j-1}) \right) / (\Delta y^2)
\]

\[
+ \left( \mu_{k+\frac{1}{2}} (u_{k+1} - u_k) - \mu_{k-\frac{1}{2}} (u_k - u_{k-1}) \right) / (\Delta z^2)
\]

\[
+ \left[ \mu_{j+\frac{1}{2}} \left( \frac{1}{4} (v_{i+1,j+1} + v_{i+1,j} - v_{i-1,j+1} - v_{i-1,j}) \right) - \mu_{j-\frac{1}{2}} \left( \frac{1}{4} (v_{i+1,j+1} + v_{i+1,j} - v_{i-1,j} - v_{i-1,j+1}) \right) \right] / (\Delta x \Delta y)
\]

\[
+ \left[ \mu_{k+\frac{1}{2}} \left( \frac{1}{4} (w_{i+1,k+1} + w_{i+1,k} - w_{i-1,k+1} - w_{i-1,k}) \right) - \mu_{k-\frac{1}{2}} \left( \frac{1}{4} (w_{i+1,k+1} + w_{i+1,k} - w_{i-1,k} - w_{i-1,k+1}) \right) \right] / (\Delta x \Delta z)
\]

And, for the \( y \)-direction, with \( \mathbf{e}_2 = (0, 1, 0) \), we have

\[
2 [\nabla \cdot (\mu \mathbf{D})] \cdot \mathbf{e}_2 = \frac{\partial}{\partial x} \left( \mu \frac{\partial v}{\partial x} + \mu \frac{\partial u}{\partial y} \right) + 2 \frac{\partial}{\partial y} \left( \mu \frac{\partial v}{\partial y} \right) + \frac{\partial}{\partial z} \left( \mu \frac{\partial v}{\partial z} + \mu \frac{\partial w}{\partial y} \right)
\]

\[
\approx \left( \mu_{i+\frac{1}{2}} (v_{i+1} - v_i) - \mu_{i-\frac{1}{2}} (v_i - v_{i-1}) \right) / (\Delta x^2)
\]

\[
+ 2 \left( \mu_{j+\frac{1}{2}} (v_{j+1} - v_j) - \mu_{j-\frac{1}{2}} (v_j - v_{j-1}) \right) / (\Delta y^2)
\]

\[
+ \left( \mu_{k+\frac{1}{2}} (v_{k+1} - v_k) - \mu_{k-\frac{1}{2}} (v_k - v_{k-1}) \right) / (\Delta z^2)
\]

\[
+ \left[ \mu_{i+\frac{1}{2}} \left( \frac{1}{4} (u_{i+1,j+1} + u_{i,j+1} - u_{i+1,j-1} - u_{i,j-1}) \right) - \mu_{i-\frac{1}{2}} \left( \frac{1}{4} (u_{i+1,j+1} + u_{i,j+1} - u_{i+1,j-1} - u_{i,j-1}) \right) \right] / (\Delta x \Delta y)
\]

\[
+ \left[ \mu_{k+\frac{1}{2}} \left( \frac{1}{4} (w_{j+1,k+1} + w_{j+1,k} - w_{j-1,k+1} - w_{j-1,k}) \right) - \mu_{k-\frac{1}{2}} \left( \frac{1}{4} (w_{j+1,k+1} + w_{j+1,k} - w_{j-1,k} - w_{j-1,k+1}) \right) \right] / (\Delta y \Delta z)
\]
For the $z$-direction, with $\mathbf{e}_3 = (0, 0, 1)$, we have

$$2 \left[ \nabla \cdot (\mu \mathbf{D}) \right] \cdot \mathbf{e}_3 = \frac{\partial}{\partial x} \left( \mu \frac{\partial w}{\partial x} + \mu \frac{\partial u}{\partial z} \right) + \frac{\partial}{\partial y} \left( \mu \frac{\partial w}{\partial y} + \mu \frac{\partial v}{\partial z} \right) + 2 \frac{\partial}{\partial z} \left( \mu \frac{\partial w}{\partial z} \right)
$$

$$\approx \left( \mu_{i+\frac{1}{2}} \left( w_{i+1} - w_i \right) - \mu_{i-\frac{1}{2}} \left( w_i - w_{i-1} \right) \right) / (\Delta x^2)
$$

$$+ \left( \mu_{j+\frac{1}{2}} \left( w_{j+1} - w_j \right) - \mu_{j-\frac{1}{2}} \left( w_j - w_{j-1} \right) \right) / (\Delta y^2)
$$

$$+ 2 \left( \mu_{k+\frac{1}{2}} \left( w_{k+1} - w_k \right) - \mu_{k-\frac{1}{2}} \left( w_k - w_{k-1} \right) \right) / (\Delta z^2)
$$

$$+ \left[ \mu_{i+\frac{1}{4}} \frac{1}{4} \left( u_{i+1,k+1} + u_{i,k+1} - u_{i+1,k-1} - u_{i,k-1} \right)
$$

$$- \mu_{i-\frac{1}{4}} \frac{1}{4} \left( u_{i,k+1} + u_{i-1,k+1} - u_{i,k-1} - u_{i-1,k-1} \right) \right] / (\Delta x \Delta z)
$$

$$+ \left[ \mu_{j+\frac{1}{4}} \frac{1}{4} \left( v_{j+1,k+1} + v_{j,k+1} - v_{j+1,k-1} - v_{j,k-1} \right)
$$

$$- \mu_{j-\frac{1}{4}} \frac{1}{4} \left( v_{j,k+1} + v_{j-1,k+1} - v_{j,k-1} - v_{j-1,k-1} \right) \right] / (\Delta y \Delta z)
$$

We can now rewrite E1/1, the momentum equation, in “ready-to-discretize” form:

$$\nabla^2 \mathbf{u}^{n+1} - \frac{\rho}{\eta_a \Delta t} \mathbf{u}^{n+1} = -\frac{1}{\eta_a} \left( \frac{\rho}{\Delta t} \mathbf{v}^n - \rho \mathbf{v}^n \cdot \nabla \mathbf{v}^n + \mathbf{R}^n \right) + \nabla s^n
$$

$$= -\frac{1}{\eta_a} \left( \frac{\rho}{\Delta t} \mathbf{v}^n - \rho \mathbf{v}^n \cdot \nabla \mathbf{v}^n \right) + \nabla s^n
$$

$$- \frac{1}{\eta_a} \mathbf{R}^n
$$

$$= -\frac{1}{\eta_a} \left( \frac{\rho}{\Delta t} \mathbf{v}^n - \rho \mathbf{v}^n \cdot \nabla \mathbf{v}^n \right) + \nabla s^n
$$

$$- \frac{1}{\eta_a} \left( \left( -\Gamma_1 \left( \frac{\partial^2 \phi_n}{\partial x^2} + \frac{\partial^2 \phi_n}{\partial y^2} + \frac{\partial^2 \phi_n}{\partial z^2} \right) + \Gamma_2 h(\phi_n) \right) \nabla \phi_n \right.
$$

$$+ 2 \nabla \cdot (\mu \mathbf{D}) - \eta_a \Delta \mathbf{v}
$$

$$= -\frac{1}{\eta_a} \left( \frac{\rho}{\Delta t} \mathbf{v}^n - \rho \mathbf{v}^n \cdot \nabla \mathbf{v}^n \right) + \nabla s^n - \Delta \mathbf{v}
$$

$$- \frac{1}{\eta_a} \left( \left( -\Gamma_1 \left( \frac{\partial^2 \phi_n}{\partial x^2} + \frac{\partial^2 \phi_n}{\partial y^2} + \frac{\partial^2 \phi_n}{\partial z^2} \right) + \Gamma_2 h(\phi_n) \right) \nabla \phi_n \right.
$$

$$+ 2 \nabla \cdot (\mu \mathbf{D}) \right)$$
A.2 EQUATION E2

We now consider the discretization of equation E2.

\[ \frac{\partial \phi_n}{\partial t} + \nabla \cdot (\phi_n \mathbf{v}) = \nabla \cdot \left( \Lambda \phi_n \nabla \frac{\delta f}{\delta \phi} \right) + g_n. \]

\[ f(\phi_n) = \frac{\Gamma_1}{2} ||\nabla \phi_n||^2 + \Gamma_2 (\phi_n^2 (1 - \phi_n)^2) \quad \text{(E2)} \]

Using the properties of the divergence operator, we can write

\[ \frac{\partial \phi_n}{\partial t} + \phi_n (\nabla \cdot \mathbf{v}) + \mathbf{v} \cdot \nabla \phi_n = \nabla \cdot \left( \Lambda \phi_n \nabla \frac{\delta f}{\delta \phi} \right) + g_n \]

The difficult part of this discretization is undoubtedly the term \( \nabla \cdot \left( \Lambda \phi_n \nabla \frac{\delta f}{\delta \phi} \right) \), which we consider below.

\[ \nabla \cdot \left( \Lambda \phi_n \nabla \frac{\delta f}{\delta \phi} \right) = \nabla \cdot \left( \Lambda \phi_n \nabla \frac{\partial}{\partial \phi_n} \left( \frac{\Gamma_1}{2} ||\nabla \phi_n||^2 + \Gamma_2 (\phi_n^2 (1 - \phi_n)^2) \right) \right) \]

\[ = \nabla \cdot \left[ \Lambda \phi_n \nabla \left( -\Gamma_1 \Delta \phi_n + \Gamma_2 (2\phi_n - 6\phi_n^2 + 4\phi_n^3) \right) \right] \]

\[ = \nabla \cdot \left[ \Lambda \phi_n \nabla \left( -\Gamma_1 \Delta \phi_n + 2\Gamma_2 \phi_n \right) \right] + \nabla \cdot \left[ \Lambda \phi_n \nabla \left( \Gamma_2 (-6\phi_n^2 + 4\phi_n^3) \right) \right] \]

\[ = -\Lambda \Gamma_1 \nabla \cdot \left[ \phi_n \nabla (\Delta \phi_n) \right] + \Lambda \Gamma_2 \nabla \cdot \left[ \phi_n \nabla \left( 2\phi_n - 6\phi_n^2 + 4\phi_n^3 \right) \right] \]

The confusing aspect of this discretization is that some of the \( \phi_n \) terms are handled implicitly while others are treated explicitly. Below, we drop the subscript \( n \) for
convenience.

\[
\nabla \cdot \left( \Lambda \phi \nabla \frac{\delta f}{\delta \phi} \right) = - \Lambda \Gamma_1 \nabla \cdot \left[ \phi^n \nabla (\Delta \phi^{n+1}) \right] \quad \text{(I1)}
\]

\[
+ \Lambda \Gamma_2 \nabla \cdot \left[ \phi^n \nabla \left( 2\phi_n - 6(\phi^n)^2 + 4(\phi^n)^3 \right) \right] \quad \text{(E1)}
\]

We now separately consider the implicit term \(I_1\) and the explicit term \(E_1\). First, \(I_1\) is discretized.

\[
- \Lambda \Gamma_1 \nabla \cdot \left[ \phi^n \nabla (\Delta \phi^{n+1}) \right] = - \Lambda \Gamma_1 \nabla \cdot \left( \phi^n \left\{ \frac{\partial}{\partial x} \Delta \phi^{n+1}, \frac{\partial}{\partial y} \Delta \phi^{n+1}, \frac{\partial}{\partial z} \Delta \phi^{n+1} \right\} \right)
\]

\[
= - \Lambda \Gamma_1 \left[ \frac{\partial}{\partial x} \left( \phi^n \frac{\partial}{\partial x} \Delta \phi^{n+1} \right) + \frac{\partial}{\partial y} \left( \phi^n \frac{\partial}{\partial y} \Delta \phi^{n+1} \right) + \frac{\partial}{\partial z} \left( \phi^n \frac{\partial}{\partial z} \Delta \phi^{n+1} \right) \right]
\]

\[
= - \Lambda \Gamma_1 \left[ \frac{1}{\Delta x} \left( \phi^n_{i+\frac{1}{2}} \frac{\partial}{\partial x} \Delta \phi^{n+1}_{i+\frac{1}{2}} - \phi^n_{i-\frac{1}{2}} \frac{\partial}{\partial x} \Delta \phi^{n+1}_{i-\frac{1}{2}} \right) \right.
\]

\[
+ \frac{1}{\Delta y} \left( \phi^n_{j+\frac{1}{2}} \frac{\partial}{\partial y} \Delta \phi^{n+1}_{j+\frac{1}{2}} - \phi^n_{j-\frac{1}{2}} \frac{\partial}{\partial y} \Delta \phi^{n+1}_{j-\frac{1}{2}} \right)
\]

\[
+ \frac{1}{\Delta z} \left( \phi^n_{k+\frac{1}{2}} \frac{\partial}{\partial z} \Delta \phi^{n+1}_{k+\frac{1}{2}} - \phi^n_{k-\frac{1}{2}} \frac{\partial}{\partial z} \Delta \phi^{n+1}_{k-\frac{1}{2}} \right) \left] \right. \right.
\]

\[
= - \Lambda \Gamma_1 \left[ \frac{1}{\Delta x^2} \left( \phi^n_{i+\frac{1}{2}} \frac{\partial}{\partial x} \Delta \phi^{n+1}_{i+\frac{1}{2}} - \phi^n_{i-\frac{1}{2}} \frac{\partial}{\partial x} \Delta \phi^{n+1}_{i-\frac{1}{2}} \right) \right.
\]

\[
+ \frac{1}{\Delta y^2} \left( \phi^n_{j+\frac{1}{2}} \frac{\partial}{\partial y} \Delta \phi^{n+1}_{j+\frac{1}{2}} - \phi^n_{j-\frac{1}{2}} \frac{\partial}{\partial y} \Delta \phi^{n+1}_{j-\frac{1}{2}} \right) \right.
\]

\[
+ \frac{1}{\Delta z^2} \left( \phi^n_{k+\frac{1}{2}} \frac{\partial}{\partial z} \Delta \phi^{n+1}_{k+\frac{1}{2}} - \phi^n_{k-\frac{1}{2}} \frac{\partial}{\partial z} \Delta \phi^{n+1}_{k-\frac{1}{2}} \right) \left] \right. \right.
\]
Notice that all the terms in the last equation involve the laplacian ($\triangle$) times some constant coefficient.

\[
A \triangle \phi^{n+1}_{i,j,k} = A \left( \frac{1}{\Delta x^2} (\phi^{n+1}_{i+1} - 2\phi^{n+1}_i + \phi^{n+1}_{i-1}) + \frac{1}{\Delta y^2} (\phi^{n+1}_{j+1} - 2\phi^{n+1}_j + \phi^{n+1}_{j-1}) + \frac{1}{\Delta z^2} (\phi^{n+1}_{k+1} - 2\phi^{n+1}_k + \phi^{n+1}_{k-1}) \right)
\]

\[
= A \left( -2 \frac{\frac{d\phi^{n+1}}{dx}}{\Delta x^2} - 2 \frac{\frac{d\phi^{n+1}}{dy}}{\Delta y^2} - 2 \frac{\frac{d\phi^{n+1}}{dz}}{\Delta z^2} \right) \phi^{n+1}_{i,j,k}
\]

Using this pattern, we can painstakingly extract the matrix coefficients. Let

\[
\overline{d\phi} = \left( \begin{array}{ccc}
-\frac{\frac{d\phi}{dx}}{\Delta x^2} & -\frac{\frac{d\phi}{dy}}{\Delta y^2} & -\frac{\frac{d\phi}{dz}}{\Delta z^2} \\
\end{array} \right)
\]

\[
\overline{d\phi} \phi^{n+1}_{i,j,k} = \frac{\frac{d\phi}{dx}}{\Delta x^2} \phi^{n+1}_{i+1,j,k} + \frac{\frac{d\phi}{dy}}{\Delta y^2} \phi^{n+1}_{i,j+1,k} + \frac{\frac{d\phi}{dz}}{\Delta z^2} \phi^{n+1}_{i,j,k+1} 
\]

\[
\phi^{n+1}_{i+\frac{1}{2},j,k} = \frac{\Delta x}{2} \phi^{n+1}_{i+1,j,k} + \frac{\phi^{n+1}_{i+\frac{1}{2},j,k+1}}{\Delta x} + \frac{\phi^{n+1}_{i+\frac{1}{2},j,k-1}}{\Delta x} + \frac{\phi^{n+1}_{i+1,j,k}}{\Delta x} 
\]

\[
\phi^{n+1}_{i,j+\frac{1}{2},k} = \frac{\Delta y}{2} \phi^{n+1}_{i,j+1,k} + \frac{\phi^{n+1}_{i,j+\frac{1}{2},k+1}}{\Delta y} + \frac{\phi^{n+1}_{i,j+\frac{1}{2},k-1}}{\Delta y} + \frac{\phi^{n+1}_{i,j+1,k}}{\Delta y} 
\]

\[
\phi^{n+1}_{i,j,k+\frac{1}{2}} = \frac{\Delta z}{2} \phi^{n+1}_{i,j,k+1} + \frac{\phi^{n+1}_{i,j+\frac{1}{2},k+1}}{\Delta z} + \frac{\phi^{n+1}_{i,j+\frac{1}{2},k-1}}{\Delta z} + \frac{\phi^{n+1}_{i,j,k+1}}{\Delta z} 
\]
\[
\frac{\phi^{n+1}}{\Delta z^2} \Delta \phi_{i,j,k+1}^{n+1} = -\frac{2\phi_{k+\frac{1}{2}}^{n+1}}{\Delta z^2 \Delta r^2} \phi_{i,j,k+1}^{n+1} + \frac{\phi_{k+\frac{1}{2}}^{n+1}}{\Delta z^2 \Delta x^2} \phi_{i+1,k+1}^{n+1} + \frac{\phi_{k+\frac{1}{2}}^{n+1}}{\Delta z^2 \Delta y^2} \phi_{j+1,k+1}^{n+1} + \frac{\phi_{k+1}^{n+1}}{\Delta z^2} \phi_{k+2}^{n+1} \\
+ \frac{\phi_{k+\frac{1}{2}}^{n+1}}{\Delta z^2 \Delta x^2} \phi_{i-1,k+1}^{n+1} + \frac{\phi_{k+\frac{1}{2}}^{n+1}}{\Delta z^2 \Delta y^2} \phi_{j-1,k+1}^{n+1} + \frac{\phi_{k+1}^{n+1}}{\Delta z^2} \phi_{k+2}^{n+1} 
\]

\[
\frac{\phi^{n-1}}{\Delta x^2} \Delta \phi_{i-1,j,k}^{n+1} = -\frac{2\phi_{i-\frac{1}{2}}^{n-1}}{\Delta x^2 \Delta r^2} \phi_{i-1,j,k}^{n+1} + \frac{\phi_{i-\frac{1}{2}}^{n-1}}{\Delta x^2} \phi_{i-1,j,k}^{n+1} + \frac{\phi_{i-\frac{1}{2}}^{n-1}}{\Delta x^2 \Delta y^2} \phi_{i-1,j+1}^{n+1} + \frac{\phi_{i-\frac{1}{2}}^{n-1}}{\Delta x^2 \Delta z^2} \phi_{i-1,k+1}^{n+1} \\
+ \frac{\phi_{i-\frac{1}{2}}^{n-1}}{\Delta x^2} \phi_{i-2,j,k}^{n+1} + \frac{\phi_{i-\frac{1}{2}}^{n-1}}{\Delta x^2 \Delta y^2} \phi_{i-1,j-1}^{n+1} + \frac{\phi_{i-\frac{1}{2}}^{n-1}}{\Delta x^2 \Delta z^2} \phi_{i-1,k-1}^{n+1} 
\]

\[
\frac{\phi^{n-1}}{\Delta y^2} \Delta \phi_{i,j-1,k}^{n+1} = -\frac{2\phi_{j-\frac{1}{2}}^{n-1}}{\Delta y^2 \Delta r^2} \phi_{i,j-1,k}^{n+1} + \frac{\phi_{j-\frac{1}{2}}^{n-1}}{\Delta y^2} \phi_{i,j-1,k}^{n+1} + \frac{\phi_{j-\frac{1}{2}}^{n-1}}{\Delta y^2 \Delta x^2} \phi_{i+1,j-1}^{n+1} + \frac{\phi_{j-\frac{1}{2}}^{n-1}}{\Delta y^2 \Delta z^2} \phi_{i-1,k+1}^{n+1} \\
+ \frac{\phi_{j-\frac{1}{2}}^{n-1}}{\Delta y^2} \phi_{j-2,k}^{n+1} + \frac{\phi_{j-\frac{1}{2}}^{n-1}}{\Delta y^2 \Delta x^2} \phi_{i-1,j-1}^{n+1} + \frac{\phi_{j-\frac{1}{2}}^{n-1}}{\Delta y^2 \Delta z^2} \phi_{j-1,k-1}^{n+1} 
\]

\[
\frac{\phi^{n-1}}{\Delta z^2} \Delta \phi_{i,j,k-1}^{n+1} = -\frac{2\phi_{k-\frac{1}{2}}^{n-1}}{\Delta z^2 \Delta r^2} \phi_{i,j,k-1}^{n+1} + \frac{\phi_{k-\frac{1}{2}}^{n-1}}{\Delta z^2 \Delta x^2} \phi_{i+1,k-1}^{n+1} + \frac{\phi_{k-\frac{1}{2}}^{n-1}}{\Delta z^2 \Delta y^2} \phi_{j+1,k-1}^{n+1} + \frac{\phi_{k-1}^{n-1}}{\Delta z^2 \Delta z^2} \phi_{k+2}^{n+1} \\
+ \frac{\phi_{k-\frac{1}{2}}^{n-1}}{\Delta z^2 \Delta x^2} \phi_{i-1,k-1}^{n+1} + \frac{\phi_{k-\frac{1}{2}}^{n-1}}{\Delta z^2 \Delta y^2} \phi_{j-1,k-1}^{n+1} + \frac{\phi_{k-1}^{n-1}}{\Delta z^2 \Delta z^2} \phi_{k+2}^{n+1} 
\]
Collecting terms from just the right-hand side of equation (E2), we see that if

$$\phi^n_x = \phi^n_{i + \frac{1}{2}} + \phi^n_{i - \frac{1}{2}}$$, then

<table>
<thead>
<tr>
<th>Node</th>
<th>Value</th>
</tr>
</thead>
</table>
| $i, j, k$ | $\Lambda_1 \left[ \phi^n_x \left( \frac{3}{\Delta x^4} + \frac{2}{\Delta x^2 \Delta y^2} + \frac{2}{\Delta x^2 \Delta z^2} \right) \right. \\
| | $\left. + \phi^n_y \left( \frac{2}{\Delta y^2 \Delta x^2} + \frac{3}{\Delta y^4} + \frac{2}{\Delta y^2 \Delta z^2} \right) \\
| | $\left. + \phi^n_z \left( \frac{2}{\Delta z^2 \Delta x^2} + \frac{2}{\Delta z^2 \Delta y^2} + \frac{3}{\Delta z^4} \right) \right]$
| $i - 1$ | $-\Lambda_1 \left[ \frac{1}{\Delta x^2} \left( \frac{3}{\Delta x^2} + \frac{2}{\Delta y^2} + \frac{2}{\Delta z^2} \right) \right. \\
| | $\left. \phi^n_{i - 1} + \phi^n_{i + \frac{1}{2}} + \phi^n_y + \phi^n_z \right]$ 
| $i + 1$ | $-\Lambda_1 \left[ \frac{1}{\Delta x^2} \left( \frac{3}{\Delta x^2} + \frac{2}{\Delta y^2} + \frac{2}{\Delta z^2} \right) \right. \\
| | $\left. \phi^n_{i + 1} + \phi^n_{i - \frac{1}{2}} + \phi^n_y + \phi^n_z \right]$ 
| $j - 1$ | $-\Lambda_1 \left[ \frac{1}{\Delta y^2} \left( \frac{2}{\Delta x^2} + \frac{3}{\Delta y^2} + \frac{2}{\Delta z^2} \right) \right. \\
| | $\left. \phi^n_{j - 1} + \phi^n_{j + \frac{1}{2}} + \phi^n_y + \phi^n_z \right]$ 
| $j + 1$ | $-\Lambda_1 \left[ \frac{1}{\Delta y^2} \left( \frac{2}{\Delta x^2} + \frac{3}{\Delta y^2} + \frac{2}{\Delta z^2} \right) \right. \\
| | $\left. \phi^n_{j + 1} + \phi^n_{j - \frac{1}{2}} + \phi^n_y + \phi^n_z \right]$ 
| $k - 1$ | $-\Lambda_1 \left[ \frac{1}{\Delta z^2} \left( \frac{2}{\Delta x^2} + \frac{3}{\Delta y^2} + \frac{2}{\Delta z^2} \right) \right. \\
| | $\left. \phi^n_{k - 1} + \phi^n_{k + \frac{1}{2}} + \phi^n_y + \phi^n_z \right]$ 
| $k + 1$ | $-\Lambda_1 \left[ \frac{1}{\Delta z^2} \left( \frac{2}{\Delta x^2} + \frac{3}{\Delta y^2} + \frac{2}{\Delta z^2} \right) \right. \\
| | $\left. \phi^n_{k + 1} + \phi^n_{k - \frac{1}{2}} + \phi^n_y + \phi^n_z \right]$ 
| $i - 2$ | $\Lambda_1 \phi^n_{i - \frac{1}{2}} / \Delta x^4$ 
| $i + 2$ | $\Lambda_1 \phi^n_{i + \frac{1}{2}} / \Delta x^4$ 
| $j - 2$ | $\Lambda_1 \phi^n_{j - \frac{1}{2}} / \Delta y^4$ 
| $j + 2$ | $\Lambda_1 \phi^n_{j + \frac{1}{2}} / \Delta y^4$ 
| $k - 2$ | $\Lambda_1 \phi^n_{k - \frac{1}{2}} / \Delta z^4$ 
| $k + 2$ | $\Lambda_1 \phi^n_{k + \frac{1}{2}} / \Delta z^4$

which concludes all the ‘unidirectional' elements.
Now for the ‘cross’ elements.

<table>
<thead>
<tr>
<th>Node</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i - 1, j - 1, k$</td>
<td>$\Lambda \Gamma_1 \left( \phi_{i-\frac{1}{2}}^n + \phi_{j-\frac{1}{2}}^n \right) / (\Delta x^2 \Delta y^2)$</td>
</tr>
<tr>
<td>$i - 1, j + 1, k$</td>
<td>$\Lambda \Gamma_1 \left( \phi_{i-\frac{1}{2}}^n + \phi_{j+\frac{1}{2}}^n \right) / (\Delta x^2 \Delta y^2)$</td>
</tr>
<tr>
<td>$i - 1, j, k - 1$</td>
<td>$\Lambda \Gamma_1 \left( \phi_{i-\frac{1}{2}}^n + \phi_{k-\frac{1}{2}}^n \right) / (\Delta x^2 \Delta z^2)$</td>
</tr>
<tr>
<td>$i - 1, j, k + 1$</td>
<td>$\Lambda \Gamma_1 \left( \phi_{i-\frac{1}{2}}^n + \phi_{k+\frac{1}{2}}^n \right) / (\Delta x^2 \Delta z^2)$</td>
</tr>
<tr>
<td>$i + 1, j - 1, k$</td>
<td>$\Lambda \Gamma_1 \left( \phi_{i+\frac{1}{2}}^n + \phi_{j-\frac{1}{2}}^n \right) / (\Delta x^2 \Delta y^2)$</td>
</tr>
<tr>
<td>$i + 1, j + 1, k$</td>
<td>$\Lambda \Gamma_1 \left( \phi_{i+\frac{1}{2}}^n + \phi_{j+\frac{1}{2}}^n \right) / (\Delta x^2 \Delta y^2)$</td>
</tr>
<tr>
<td>$i + 1, j, k - 1$</td>
<td>$\Lambda \Gamma_1 \left( \phi_{i+\frac{1}{2}}^n + \phi_{k-\frac{1}{2}}^n \right) / (\Delta x^2 \Delta z^2)$</td>
</tr>
<tr>
<td>$i + 1, j, k + 1$</td>
<td>$\Lambda \Gamma_1 \left( \phi_{i+\frac{1}{2}}^n + \phi_{k+\frac{1}{2}}^n \right) / (\Delta x^2 \Delta z^2)$</td>
</tr>
<tr>
<td>$i, j - 1, k - 1$</td>
<td>$\Lambda \Gamma_1 \left( \phi_{j-\frac{1}{2}}^n + \phi_{k-\frac{1}{2}}^n \right) / (\Delta y^2 \Delta z^2)$</td>
</tr>
<tr>
<td>$i, j - 1, k + 1$</td>
<td>$\Lambda \Gamma_1 \left( \phi_{j-\frac{1}{2}}^n + \phi_{k+\frac{1}{2}}^n \right) / (\Delta y^2 \Delta z^2)$</td>
</tr>
<tr>
<td>$i, j + 1, k - 1$</td>
<td>$\Lambda \Gamma_1 \left( \phi_{j+\frac{1}{2}}^n + \phi_{k-\frac{1}{2}}^n \right) / (\Delta y^2 \Delta z^2)$</td>
</tr>
<tr>
<td>$i, j + 1, k + 1$</td>
<td>$\Lambda \Gamma_1 \left( \phi_{j+\frac{1}{2}}^n + \phi_{k+\frac{1}{2}}^n \right) / (\Delta y^2 \Delta z^2)$</td>
</tr>
</tbody>
</table>

A.3 EQUATION E3

We now consider the discretization of nutrient transport equation E3.

\[
\frac{\partial}{\partial t} (\phi_s c) + \nabla \cdot (\phi_s c \mathbf{v} - D_s \phi_s \nabla c) = -g_c \tag{E3}
\]

\[
\frac{\phi_s^{n+1} c^{n+1} - \phi_s^n c^n}{\Delta t} + \nabla \cdot (\phi_s^{n+1} c^{n+1} \mathbf{v}^{n+1}) - D_s \nabla \cdot (\phi_s^{n+1} \nabla c^{n+1}) = -g_c
\]

\[
\frac{\phi_s^{n+1} c^{n+1} - \phi_s^n c^n}{\Delta t} + \mathbf{v}^{n+1} \cdot \nabla (\phi_s^{n+1} c^{n+1}) - D_s \nabla \cdot (\phi_s^{n+1} \nabla c^{n+1}) = -g_c
\]

Above, we have used $\nabla \cdot \mathbf{v}^{n+1} = 0$, since the algorithm ensures the new velocity is divergence free. Once again, each term is discretized separately. First, the advection term.
\( \mathbf{v}^{n+1} \cdot \nabla (\phi_s^{n+1} c^{n+1}) = u^{n+1} \frac{\partial}{\partial x} (\phi_s^{n+1} c^{n+1}) + v^{n+1} \frac{\partial}{\partial y} (\phi_s^{n+1} c^{n+1}) + w^{n+1} \frac{\partial}{\partial z} (\phi_s^{n+1} c^{n+1}) \\
= u_i^{n+1} \frac{\phi_s^{n+1} c_{i+1}^{n+1} - \phi_s^{n+1} c_{i-1}^{n+1}}{2 \Delta x} + v_j^{n+1} \frac{\phi_s^{n+1} c_{j+1}^{n+1} - \phi_s^{n+1} c_{j-1}^{n+1}}{2 \Delta y} + w_k^{n+1} \frac{\phi_s^{n+1} c_{k+1}^{n+1} - \phi_s^{n+1} c_{k-1}^{n+1}}{2 \Delta z} \\
= \frac{u_i^{n+1}}{2 \Delta x} \phi_{s,i+1}^{n+1} c_{i+1}^{n+1} + \frac{v_j^{n+1}}{2 \Delta y} \phi_{s,j+1}^{n+1} c_{j+1}^{n+1} + \frac{w_k^{n+1}}{2 \Delta z} \phi_{s,k+1}^{n+1} c_{k+1}^{n+1} \\
- \frac{u_i^{n+1}}{2 \Delta x} \phi_{s,i-1}^{n+1} c_{i-1}^{n+1} - \frac{v_j^{n+1}}{2 \Delta y} \phi_{s,j-1}^{n+1} c_{j-1}^{n+1} - \frac{w_k^{n+1}}{2 \Delta z} \phi_{s,k-1}^{n+1} c_{k-1}^{n+1} \\
\\Now the diffusion term. \\
- D_s \nabla \cdot (\phi_s^{n+1} \nabla c^{n+1}) = - D_s \nabla \cdot \left( \phi_s^{n+1} \left\{ \frac{\partial}{\partial x} c^{n+1}, \frac{\partial}{\partial y} c^{n+1}, \frac{\partial}{\partial z} c^{n+1} \right\} \right) \\
= - D_s \left( \frac{\partial}{\partial x} \left( \phi_s^{n+1} \frac{\partial}{\partial x} c^{n+1} \right) + \frac{\partial}{\partial y} \left( \phi_s^{n+1} \frac{\partial}{\partial y} c^{n+1} \right) + \frac{\partial}{\partial z} \left( \phi_s^{n+1} \frac{\partial}{\partial z} c^{n+1} \right) \right) \\
= - D_s \left( \frac{1}{\Delta x^2} \left( \phi_s^{n+1} c_{i+1}^{n+1} - \phi_s^{n+1} c_{i-1}^{n+1} \right) + \frac{1}{\Delta y^2} \left( \phi_s^{n+1} c_{j+1}^{n+1} - \phi_s^{n+1} c_{j-1}^{n+1} \right) + \frac{1}{\Delta z^2} \left( \phi_s^{n+1} c_{k+1}^{n+1} - \phi_s^{n+1} c_{k-1}^{n+1} \right) \right)