Scalable Solvers for Multicomponent Flows

Kars Knook [1]

Patrick Farrell [1 2]

[1]University of Oxford
[2]Charles University

February, 2025

Modeling each fluid component of a mixture is essential in a wide range of applications in the chemical industry, geosciences, combustion, battery modeling, separation processes, etc.

Modeling each fluid component of a mixture is essential in a wide range of applications in the chemical industry, geosciences, combustion, battery modeling, separation processes, etc.

Unfortunately, numerical computation of the velocity and concentration of every component, as well as other values of interest such as temperature and pressure, becomes expensive quickly.

Modeling each fluid component of a mixture is essential in a wide range of applications in the chemical industry, geosciences, combustion, battery modeling, separation processes, etc.

Unfortunately, numerical computation of the velocity and concentration of every component, as well as other values of interest such as temperature and pressure, becomes expensive quickly.

Our goal

Develop finite element discretisations and scalable solvers for a wide variety of multicomponent flows.

Modeling each fluid component of a mixture is essential in a wide range of applications in the chemical industry, geosciences, combustion, battery modeling, separation processes, etc.

Unfortunately, numerical computation of the velocity and concentration of every component, as well as other values of interest such as temperature and pressure, becomes expensive quickly.

Our goal

Develop finite element discretisations and scalable solvers for a wide variety of multicomponent flows.

We would like to account for **diffusive, convective, non-ideal mixing, thermal, pressure and electrochemical effects** for **steady and transient** multicomponent flows.

Modeling each fluid component of a mixture is essential in a wide range of applications in the chemical industry, geosciences, combustion, battery modeling, separation processes, etc.

Unfortunately, numerical computation of the velocity and concentration of every component, as well as other values of interest such as temperature and pressure, becomes expensive quickly.

Our goal

Develop finite element discretisations and scalable solvers for a wide variety of multicomponent flows.

We would like to account for **diffusive, convective, non-ideal mixing, thermal, pressure and electrochemical effects** for **steady and transient** multicomponent flows.

Scalable solvers: parallelisable on modern computer hardware.

Section 2

Onsager–Stefan–Maxwell equations

Let's start with the simplest case of the OSM equations: **multicomponent diffusion of ideal gaseous mixtures**. Furthermore, we assume isothermal, isobaric, and steady-state conditions.

Let's start with the simplest case of the OSM equations: **multicomponent diffusion of ideal gaseous mixtures**. Furthermore, we assume isothermal, isobaric, and steady-state conditions.

Steady, isothermal, isobaric, and ideal gaseous OSM problem

For given v_{bulk} and $\{r_i\}_{i=1}^n$, find $\{c_i\}_{i=1}^n$ and $\{v_i\}_{i=1}^n$ such that

$$-c_i \nabla \mu_i = \sum_j \boldsymbol{M}_{ij} v_j \qquad \forall i \in \{1, \dots, n\}, \qquad (\star)$$

$$\nabla \cdot (c_i v_i) = r_i \qquad \forall i \in \{1, \dots, n\},$$

$$v_{\text{bulk}} = \frac{\sum_i M_i c_i v_i}{\rho}, \qquad (\star\star)$$

where $\rho := \sum_{i=1}^n M_i c_i$ is the density and $\boldsymbol{M}$ is the Onsager transport matrix defined as

$$\boldsymbol{M}_{ij} = \begin{cases} -\frac{RTc_ic_j}{\mathcal{D}_{ij}c_T} & \text{if } i \neq j, \\ \sum_{k \neq i}^n \frac{RTc_ic_k}{\mathcal{D}_{ik}c_T} & \text{if } i = j. \end{cases}$$

We add the mass-average constraint $(\star\star)$ to $(\star)$ as an augmentation term, and rewrite the OSM equations in terms of the mass fluxes $\{J_i\}_{i=1}^n$ and chemical potentials $\{\mu_i\}_{i=1}^n$

$$\begin{aligned} J_i &= M_i c_i v_i && \forall i \in \{1, \ldots, n\}, \\ \mu_i &= e^{c_i} && \forall i \in \{1, \ldots, n\}. \end{aligned}$$

We add the mass-average constraint $(\star\star)$ to $(\star)$ as an augmentation term, and rewrite the OSM equations in terms of the mass fluxes $\{J_i\}_{i=1}^n$ and chemical potentials $\{\mu_i\}_{i=1}^n$

$$
\begin{aligned}
J_i &= M_i c_i v_i \qquad && \forall i \in \{1, \dots, n\}, \\
\mu_i &= e^{c_i} \qquad && \forall i \in \{1, \dots, n\}.
\end{aligned}
$$

We can derive a weak formulation with $J_i \in H(\mathrm{div})$ and $\mu_i \in L^2$ for each i, and discretise using conforming FEM with arbitrary polynomial degree.

For a **Picard linearisation** of the OSM problem we can prove continuous well-posedness and discrete well-posedness and quasi-optimality. Empirically, the Picard linearisation can be solved at each step of a fixed point iteration to obtain a solution to the fully nonlinear problem.

In practice, we employ **Newton's method** because of its superior convergence.

At each step of Newton's method we have to solve a linear system.

At each step of Newton's method we have to solve a linear system. We can do this with Gaussian elimination, but the computational complexity is $\mathcal{O}(N^3)$ and we would also quickly run out of RAM. The same problems persist for Gaussian elimination for sparse linear systems.

At each step of Newton's method we have to solve a linear system. We can do this with Gaussian elimination, but the computational complexity is $\mathcal{O}(N^3)$ and we would also quickly run out of RAM. The same problems persist for Gaussian elimination for sparse linear systems.

Hence, we need to use an iterative solver.

At each step of Newton's method we have to solve a linear system. We can do this with Gaussian elimination, but the computational complexity is $\mathcal{O}(N^3)$ and we would also quickly run out of RAM. The same problems persist for Gaussian elimination for sparse linear systems.

Hence, we need to use an iterative solver. We start with a stationary iterative method for $Ax = b$ with splitting $A = M - N$

$$Mx^{k+1} = Nx^k + b.$$

where M is usually chosen to resemble A but easier to solve.

At each step of Newton's method we have to solve a linear system. We can do this with Gaussian elimination, but the computational complexity is $\mathcal{O}(N^3)$ and we would also quickly run out of RAM. The same problems persist for Gaussian elimination for sparse linear systems.

Hence, we need to use an iterative solver. We start with a stationary iterative method for $Ax = b$ with splitting $A = M - N$

$$Mx^{k+1} = Nx^k + b.$$

where M is usually chosen to resemble A but easier to solve.

The stationary iterative method can be improved by the Generalised Minimum RESidual method (GMRES) which finds $y^{k+1} \in \text{span}(\{x^0, \ldots, x^{k+1}\})$ such that the residual $\|b - Ay^{k+1}\|_2$ is minimised.

In this setting, GMRES is referred to as the iterative solver and the stationary iterative method is the preconditioner.

At each step of Newton's method we have to solve a linear system. We can do this with Gaussian elimination, but the computational complexity is $\mathcal{O}(N^3)$ and we would also quickly run out of RAM. The same problems persist for Gaussian elimination for sparse linear systems.

Hence, we need to use an iterative solver. We start with a stationary iterative method for $Ax = b$ with splitting $A = M - N$

$$Mx^{k+1} = Nx^k + b.$$

where M is usually chosen to resemble A but easier to solve.

The stationary iterative method can be improved by the Generalised Minimum RESidual method (GMRES) which finds $y^{k+1} \in \text{span}(\{x^0, \ldots, x^{k+1}\})$ such that the residual $\|b - Ay^{k+1}\|_2$ is minimised.

In this setting, GMRES is referred to as the iterative solver and the stationary iterative method is the preconditioner. **This talk is essentially about good choices for M.**

At each step of Newton's method we have to solve a linear system with the Jacobian. To determine a good choice for M we should look at the matrix structure of the Jacobian

$$\mathcal{J} = \begin{array}{c c} & \begin{array}{cc} \bar{J} & \bar{\mu} \end{array} \\ \begin{array}{c} \bar{\tau} \\ \bar{w} \end{array} & \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & \end{pmatrix} \end{array},$$

where we have used bar notation to denote n-tuples, e.g. $\bar{J} = (J_1, \ldots, J_n)$, and $\bar{\tau}$ and $\bar{w}$ are corresponding test functions.

At each step of Newton's method we have to solve a linear system with the Jacobian. To determine a good choice for M we should look at the matrix structure of the Jacobian

$$\mathcal{J} = \begin{array}{c c} & \begin{array}{cc} \bar{J} & \bar{\mu} \end{array} \\ \begin{array}{c} \bar{\tau} \\ \bar{w} \end{array} & \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & \end{pmatrix} \end{array},$$

where we have used bar notation to denote n-tuples, e.g. $\bar{J} = (J_1, \ldots, J_n)$, and $\bar{\tau}$ and $\bar{w}$ are corresponding test functions. We will explore two preconditioning approaches:

- monolithic preconditioner, i.e. M considers $\mathcal{J}$ completely,
- block preconditioner, i.e. $M = \begin{pmatrix} D_1 & \\ & D_2 \end{pmatrix}$ and D_1, D_2 are solved iteratively.

At each step of Newton's method we have to solve a linear system with the Jacobian. To determine a good choice for M we should look at the matrix structure of the Jacobian

$$\mathcal{J} = \begin{array}{c c} & \begin{array}{cc} \bar{J} & \bar{\mu} \end{array} \\ \begin{array}{c} \bar{\tau} \\ \bar{w} \end{array} & \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & \end{pmatrix} \end{array},$$

where we have used bar notation to denote n-tuples, e.g. $\bar{J} = (J_1, \ldots, J_n)$, and $\bar{\tau}$ and $\bar{w}$ are corresponding test functions. We will explore two preconditioning approaches:

- monolithic preconditioner, i.e. M considers $\mathcal{J}$ completely,
- block preconditioner, i.e. $M = \begin{pmatrix} D_1 & \\ & D_2 \end{pmatrix}$ and D_1, D_2 are solved iteratively.

Goal

The preconditioner exhibits bounded GMRES iterations for arbitrary mesh refinements and polynomial degree, as well as parallelisation of its most costly operation.

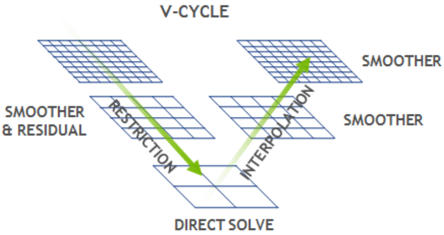
Section 3

Monolithic multigrid preconditioner

Elliptic problems: given a current iterate x^k for $Ax = b$, the simplest stationary iterative methods, e.g. Richardson, Jacobi, Gauss–Seidel iteration, only reduce the frequencies similar to the mesh size that are present in the error.

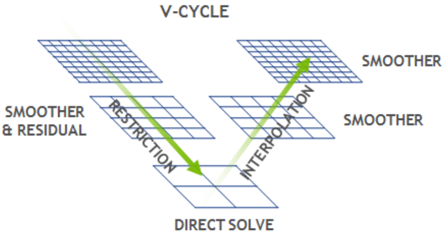
Elliptic problems: given a current iterate x^k for $Ax = b$, the simplest stationary iterative methods, e.g. Richardson, Jacobi, Gauss–Seidel iteration, only reduce the frequencies similar to the mesh size that are present in the error.

Multigrid moves the iterate between meshes with different refinements to reduce all frequencies.



Elliptic problems: given a current iterate x^k for $Ax = b$, the simplest stationary iterative methods, e.g. Richardson, Jacobi, Gauss–Seidel iteration, only reduce the frequencies similar to the mesh size that are present in the error.

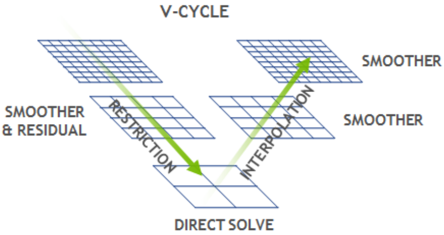
Multigrid moves the iterate between meshes with different refinements to reduce all frequencies.



Often, Gaussian elimination is used on the coarsest mesh and a parallel stationary iterative method on the finer meshes.

Elliptic problems: given a current iterate x^k for $Ax = b$, the simplest stationary iterative methods, e.g. Richardson, Jacobi, Gauss–Seidel iteration, only reduce the frequencies similar to the mesh size that are present in the error.

Multigrid moves the iterate between meshes with different refinements to reduce all frequencies.

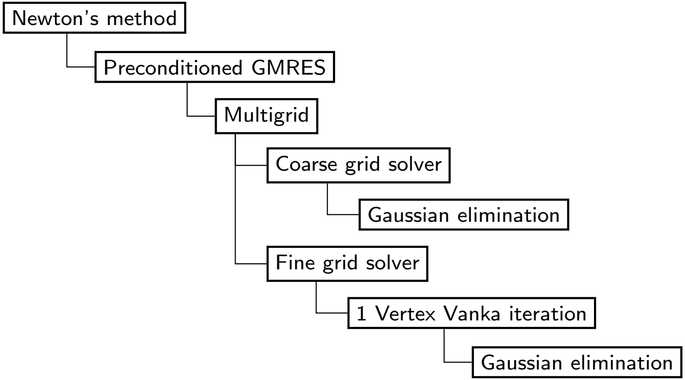


Often, Gaussian elimination is used on the coarsest mesh and a parallel stationary iterative method on the finer meshes. **Empirically multigrid has proven to be a powerful solver for a wide variety of problems.**

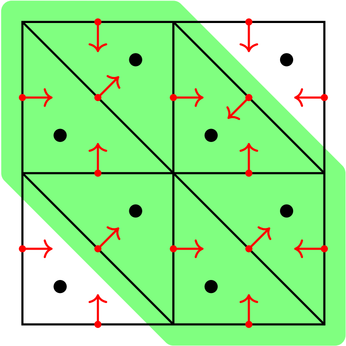
Solver diagram for the monolithic multigrid preconditioner.

- Newton's method
  - Preconditioned GMRES
    - Multigrid
      - Coarse grid solver
        - Gaussian elimination
      - Fine grid solver
        - 1 Vertex Vanka iteration
          - Gaussian elimination

Solver diagram for the monolithic multigrid preconditioner.



Vertex Vanka patch for RT_1-DG_0 discretisation.



Manufactured test problems: preconditioned GMRES iteration counts for various mesh refinements and polynomial degrees.

refinement\degree	1	2	3	4
0	1.0	1.0	1.0	1.0
1	10.0	7.0	6.5	7.0
2	11.67	7.33	7.0	7.0
3	12.0	6.5	7.5	7.0

Coarse 4×4 unit square mesh.

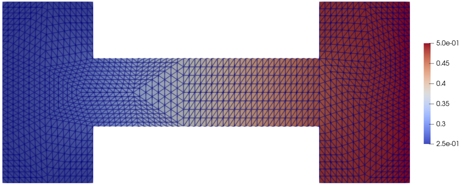
Manufactured test problems: preconditioned GMRES iteration counts for various mesh refinements and polynomial degrees.

refinement\degree	1	2	3	4
0	1.0	1.0	1.0	1.0
1	10.0	7.0	6.5	7.0
2	11.67	7.33	7.0	7.0
3	12.0	6.5	7.5	7.0

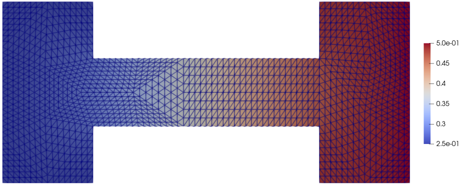
Coarse 4×4 unit square mesh.

refinement\degree	1	2	3	4
0	1.0	1.0	1.0	1.0
1	8.67	6.67	6.0	6.0
2	11.33	7.5	7.5	

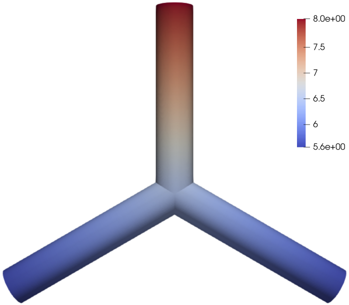
Coarse $4 \times 4 \times 4$ unit cube mesh.



Gas separation chamber: mole fraction of helium in ternary mixture of He-Ar-Kr in the presence of a temperature gradient.



Gas separation chamber: mole fraction of helium in ternary mixture of He-Ar-Kr in the presence of a temperature gradient.



Human airways: concentration of oxygen in mixture of H_2O-O2-CO_2-N_2.

Section 4

Augmented Lagrangian block preconditioner

The augmented Lagrangian preconditioning approach adds $\kappa(\nabla \cdot (c_i v_i) - r_i, \nabla \cdot \tau_i)$ where $\kappa > 0$ to the weak formulation of $(\star)$. If $\nabla \cdot (c_i v_i) = r_i$ can be satisfied exactly in the finite element space for each i, e.g. when $r_i = 0$ for each i, then the solution doesn't change.

The augmented Lagrangian preconditioning approach adds $\kappa(\nabla \cdot (c_i v_i) - r_i, \nabla \cdot \tau_i)$ where $\kappa > 0$ to the weak formulation of $(\star)$. If $\nabla \cdot (c_i v_i) = r_i$ can be satisfied exactly in the finite element space for each i, e.g. when $r_i = 0$ for each i, then the solution doesn't change.

However, the Jacobian changes to

$$\mathcal{J} = \begin{array}{c c} & \begin{array}{cc} \bar{J} & \bar{\mu} \end{array} \\ \begin{array}{c} \bar{\tau} \\ \bar{w} \end{array} & \begin{pmatrix} A_{00}^{\kappa} & A_{01} \\ A_{10} & \end{pmatrix} \end{array} \approx \begin{array}{c c} & \begin{array}{cc} \bar{J} & \bar{\mu} \end{array} \\ \begin{array}{c} \bar{\tau} \\ \bar{w} \end{array} & \begin{pmatrix} \kappa(\nabla \cdot \bar{J}, \nabla \cdot \bar{\tau}) & (\bar{\mu}, \nabla \cdot \bar{\tau}) \\ (\nabla \cdot \bar{J}, \bar{w}) & \end{pmatrix} \end{array}.$$

For large κ the augmented Lagrangian term dominates the top left block of the Jacobian.

The augmented Lagrangian preconditioning approach adds $\kappa(\nabla \cdot (c_i v_i) - r_i, \nabla \cdot \tau_i)$ where $\kappa > 0$ to the weak formulation of $(\star)$. If $\nabla \cdot (c_i v_i) = r_i$ can be satisfied exactly in the finite element space for each i, e.g. when $r_i = 0$ for each i, then the solution doesn't change.

However, the Jacobian changes to

$$\mathcal{J} = \begin{array}{c c} & \begin{array}{cc} \bar{J} & \bar{\mu} \end{array} \\ \begin{array}{c} \bar{\tau} \\ \bar{w} \end{array} & \begin{pmatrix} A_{00}^{\kappa} & A_{01} \\ A_{10} & \end{pmatrix} \end{array} \approx \begin{array}{c c} & \begin{array}{cc} \bar{J} & \bar{\mu} \end{array} \\ \begin{array}{c} \bar{\tau} \\ \bar{w} \end{array} & \begin{pmatrix} \kappa(\nabla \cdot \bar{J}, \nabla \cdot \bar{\tau}) & (\bar{\mu}, \nabla \cdot \bar{\tau}) \\ (\nabla \cdot \bar{J}, \bar{w}) & \end{pmatrix} \end{array}.$$

For large κ the augmented Lagrangian term dominates the top left block of the Jacobian. It can be shown that a good approximation for the Schur complement is $\tilde{S} = (\bar{\mu}, \bar{w})$. Hence we will use the preconditioner

$$M = \begin{pmatrix} A_{00}^{\kappa} & \\ & \hat{S} \end{pmatrix}.$$

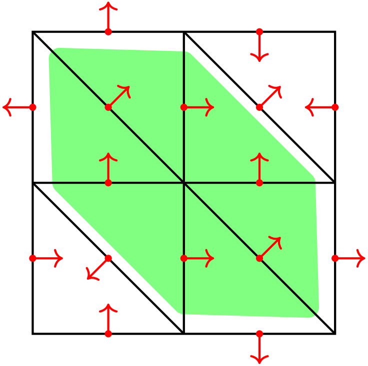
Solver diagram for the augmented Lagrangian preconditioner.

- Newton's method
  - Preconditioned GMRES
    - Block preconditioner
      - Multigrid on A_{00}^{κ} block
        - Coarse grid solver
          - Gaussian elimination
        - Fine grid solver
          - 1 Vertex star iteration
            - Gaussian elimination
      - $\hat{S}$ block
        - Gaussian elimination

Solver diagram for the augmented Lagrangian preconditioner.

- Newton's method
  - Preconditioned GMRES
    - Block preconditioner
      - Multigrid on A_{00}^{κ} block
        - Coarse grid solver
          - Gaussian elimination
        - Fine grid solver
          - 1 Vertex star iteration
            - Gaussian elimination
      - $\hat{S}$ block
        - Gaussian elimination

Vertex star patch for RT_1 discretisation.



Manufactured test problems: preconditioned GMRES iteration counts for various mesh refinements and polynomial degrees. $\kappa = 10$.

refinements \degree	1	2	3	4
1	9.75	9.75	9.75	9.75
2	20.67	20.5	23.5	20.5
3	21.67	19.0	23.0	19.0
4	20.5	18.5	20.5	18.0

Coarse 4×4 unit square mesh.

Manufactured test problems: preconditioned GMRES iteration counts for various mesh refinements and polynomial degrees. $\kappa = 10$.

refinements \degree	1	2	3	4
1	9.75	9.75	9.75	9.75
2	20.67	20.5	23.5	20.5
3	21.67	19.0	23.0	19.0
4	20.5	18.5	20.5	18.0

Coarse 4×4 unit square mesh.

refinements \degree	1	2	3	4
0	10.0	20.5	24.5	28.25
1	47.4	30.33	25.33	22.0
2	41.5	26.0	25.33	

Coarse $4 \times 4 \times 4$ unit cube mesh.

Section 5

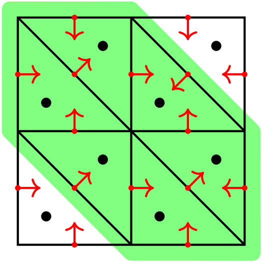
Conclusions

We have two scalable solvers for multicomponent diffusion of ideal gaseous mixtures.

We have two scalable solvers for multicomponent diffusion of ideal gaseous mixtures.

Monolithic multigrid preconditioner:

- large patch problems,
- straightforward implementation.



Vertex Vanka patch for RT_1-DG_0 discretisation.

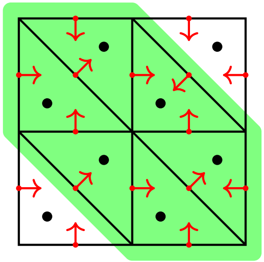
We have two scalable solvers for multicomponent diffusion of ideal gaseous mixtures.

Monolithic multigrid preconditioner:

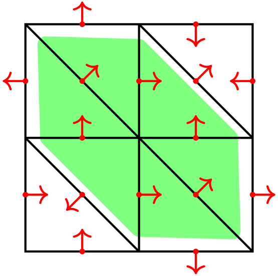
- large patch problems,
- straightforward implementation.

Augmented Lagrangian preconditioner:

- small patch problems,
- more challenging implementation.



Vertex Vanka patch for RT_1-DG_0 discretisation.

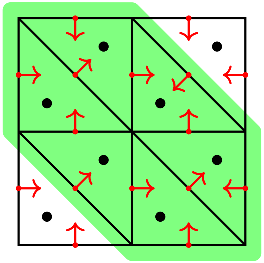


Vertex star patch for RT_1 discretisation.

We have two scalable solvers for multicomponent diffusion of ideal gaseous mixtures.

Monolithic multigrid preconditioner:

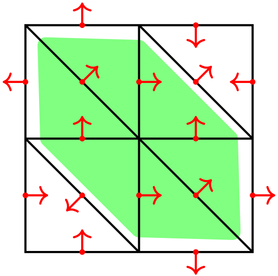
- large patch problems,
- straightforward implementation.



Vertex Vanka patch for RT_1-DG_0 discretisation.

Augmented Lagrangian preconditioner:

- small patch problems,
- more challenging implementation.



Vertex star patch for RT_1 discretisation.

In the future we hope to extend these preconditioners to more multicomponent problems!

Scalable Solvers for Multicomponent Flows

Thank you for listening!