Non-Intrusive Parallel-in-Time Solvers for Challenging Problems

Research in Industry Projects for Students - Midterm Presentation

Tran Duy Anh Le[1], Bryan Li[2], Margaret Luo[3], Daniel Agraz Vallejo[4]

Academic Mentor: Jean-Michel Maldague[5]
Industry Mentors: Robert Falgout[6], Rui Peng Li[6], Wayne Mitchell[6], Daniel Osei-Kuffuor[6]

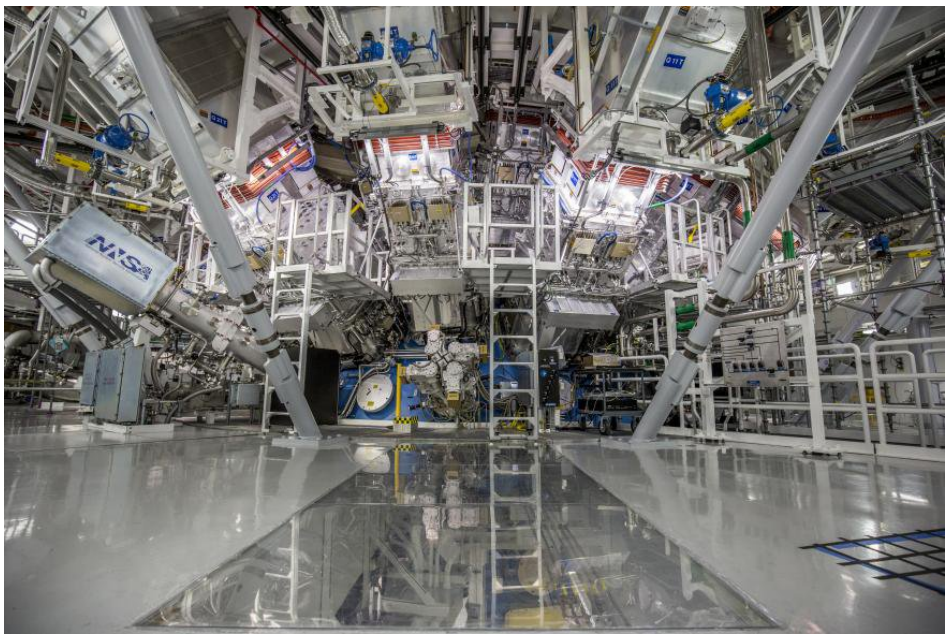
[1]University of Rochester, [2]University of California, Berkeley, [3]University of California, San Diego, [4]CETYS University

[5]University of California, Los Angeles, [6]Lawrence Livermore National Laboratory

July 24, 2024

Lawrence Livermore National Laboratory (LLNL)

Lawrence Livermore National Laboratory (LLNL)



LLNL's Center for Applied Scientific Computing

CASC
Center for Applied Scientific Computing

- Computational Physics Group
- Data Science & Analytics Group
- High Performance Computing Group
- Informatics Group
- Machine Intelligence Group
- Mathematical Algorithms & Computing Group
- Numerical Analysis & Simulations Group
- Parallel Systems Group
- Research Software Engineering Group
- Scientific Computing Group
- UQ Optimization Group

Motivation

- Processor clock speeds are stagnant, while the number of cores is increasing.
- Need arises for parallel algorithms to take advantage of increasing cores.

Motivation

- Processor clock speeds are stagnant, while the number of cores is increasing.
- Need arises for parallel algorithms to take advantage of increasing cores.

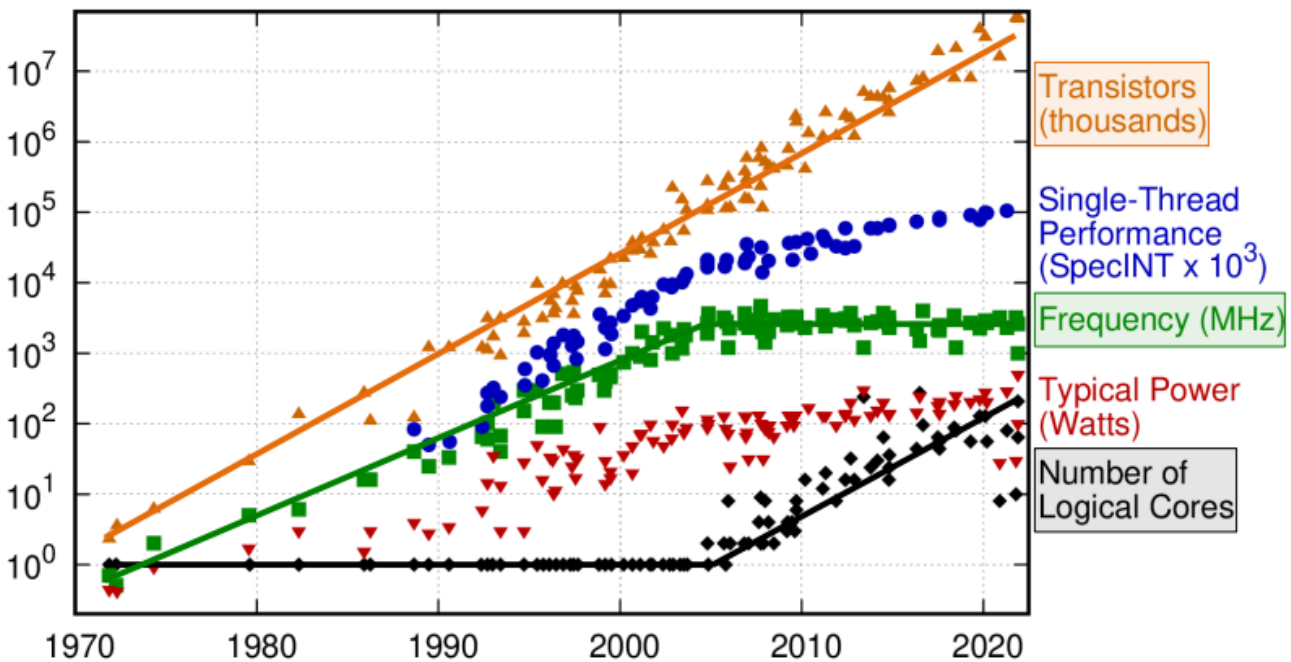


Figure: Stagnation of processor speed

What are Intrusive and Non-Intrusive Algorihms?

Intrusive:

- Parallel time stepping
  has to be explicitly coded
- Offer faster convergence
  to a specific problem

What are Intrusive and Non-Intrusive Algorihms?

Intrusive:

- Parallel time stepping has to be explicitly coded
- Offer faster convergence to a specific problem

Non-Intrusive:

- Very little effort and code are required
- Meant to be applied for a wide range of PDE problems

The Diffusion (Heat) Equation

The general form of the 1D diffusion partial differential equation (PDE) is given by:

$$u_t = a u_{xx}$$

The Diffusion (Heat) Equation

The general form of the 1D diffusion partial differential equation (PDE) is given by:

$$u_t = a u_{xx}$$

where:

- $u(x, t)$ is the concentration of the diffusing substance at position x and time t.
- a is the diffusion coefficient, which is a measure of how quickly the heat gets distributed along a surface.

The Diffusion (Heat) Equation

The general form of the 1D diffusion partial differential equation (PDE) is given by:

$$u_t = a u_{xx}$$

where:

- $u(x, t)$ is the concentration of the diffusing substance at position x and time t.
- a is the diffusion coefficient, which is a measure of how quickly the heat gets distributed along a surface.
- $u_t = \frac{\partial u}{\partial t}$ represents the rate of change of concentration with respect to time.
- $u_{xx} = \frac{\partial^2 u}{\partial x^2}$ is the diffusion term.

Advection Equation

A form of the advection equation in one spatial dimension is:

$$u_t + cu_x = 0$$

Advection Equation

A form of the advection equation in one spatial dimension is:

$$u_t + cu_x = 0$$

where:

- $u(x, t)$ is the quantity being advected at position x and time t.
- c is the constant advection velocity, representing the speed at which the substance is being transported.

Discretization using finite difference of steady-state diffusion

Consider the steady-state diffusion equation:

$$u_{xx} = f$$

where $f = f(x)$ is known, $u = u(x)$ is unknown, and $0 \leq x \leq 1$.

Discretization using finite difference of steady-state diffusion

Consider the steady-state diffusion equation:

$$u_{xx} = f$$

where $f = f(x)$ is known, $u = u(x)$ is unknown, and $0 \leq x \leq 1$. Let

$$x_i = hi, \quad h = \frac{1}{M}$$

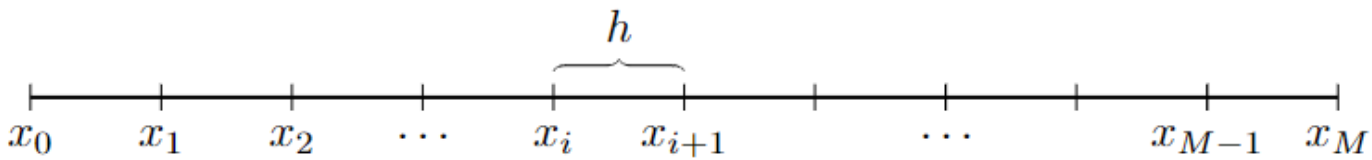


Figure: Discretization of domain

where M is the number of partitions in space.

Discretization using finite difference (continued)

$$u_{xx} = f$$

Let $u_i \approx u(x_i)$.
Using the finite difference approximation

$$u_{xx} \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}.$$

Discretization using finite difference (continued)

$$u_{xx} = f$$

Let $u_i \approx u(x_i)$.
Using the finite difference approximation

$$u_{xx} \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}.$$

we can write this in a linear system $A\mathbf{u} = \mathbf{f}$:

$$\frac{1}{h^2}\begin{pmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & & \vdots \\ 0 & 1 & \ddots & & 0 \\ \vdots & & & & 1 \\ 0 & \cdots & 0 & 1 & -2 \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_M \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_M \end{pmatrix}.$$

Relaxation with splitting

Given system $A\mathbf{x} = \mathbf{b}$
The general form for relaxation using splitting:

$$A = M + N$$

$$(M + N)\mathbf{x} = \mathbf{b}$$

$$M\mathbf{x}_{k+1} + N\mathbf{x}_k = \mathbf{b}$$

$$\begin{aligned}\mathbf{x}_{k+1} &= M^{-1}(\mathbf{b} - N\mathbf{x}_k) \\ &= \mathbf{x}_k + M^{-1}(\mathbf{b} - A\mathbf{x}_k)\end{aligned}$$

$\mathbf{b} - A\mathbf{x}_k$ is also known as the residual.

Relaxation with splitting: Jacobi and Gauss-Seidel

$$\mathbf{x}_{k+1} = \mathbf{x}_k + M^{-1}(\mathbf{b} - A\mathbf{x}_k), \quad A = M + N$$

Let $A = L + D + U$, where L is lower triangular, D is diagonal, and U is upper triangular.

Relaxation with splitting: Jacobi and Gauss-Seidel

$$\mathbf{x}_{k+1} = \mathbf{x}_k + M^{-1}(\mathbf{b} - A\mathbf{x}_k), \quad A = M + N$$

Let $A = L + D + U$, where L is lower triangular, D is diagonal, and U is upper triangular.

- In Jacobi, $M = D$ and we have

$$\mathbf{x}_{k+1} = \mathbf{x}_k + D^{-1}(\mathbf{b} - A\mathbf{x}_k).$$

Relaxation with splitting: Jacobi and Gauss-Seidel

$$\mathbf{x}_{k+1} = \mathbf{x}_k + M^{-1}(\mathbf{b} - A\mathbf{x}_k), \quad A = M + N$$

Let $A = L + D + U$, where L is lower triangular, D is diagonal, and U is upper triangular.

- In Jacobi, $M = D$ and we have

$$\mathbf{x}_{k+1} = \mathbf{x}_k + D^{-1}(\mathbf{b} - A\mathbf{x}_k).$$

- In Gauss-Seidel, $M = L + D$ and we have

$$\mathbf{x}_{k+1} = \mathbf{x}_k + (L + D)^{-1}(\mathbf{b} - A\mathbf{x}_k).$$

Why use relaxation?

Relaxation with splitting: Jacobi and Gauss-Seidel

$$\mathbf{x}_{k+1} = \mathbf{x}_k + M^{-1}(\mathbf{b} - A\mathbf{x}_k), \quad A = M + N$$

Let $A = L + D + U$, where L is lower triangular, D is diagonal, and U is upper triangular.

- In Jacobi, $M = D$ and we have

$$\mathbf{x}_{k+1} = \mathbf{x}_k + D^{-1}(\mathbf{b} - A\mathbf{x}_k).$$

- In Gauss-Seidel, $M = L + D$ and we have

$$\mathbf{x}_{k+1} = \mathbf{x}_k + (L + D)^{-1}(\mathbf{b} - A\mathbf{x}_k).$$

Why use relaxation? $O(n^3)$ to invert a matrix, but each iteration of relaxation is $O(n^2)$

Relaxation with splitting: Error Analysis

$$\mathbf{x}_{k+1} = \mathbf{x}_k + M^{-1}(\mathbf{b} - A\mathbf{x}_k)$$

Jacobi and Gauss-Seidel doesn't converge for all matrices.

Relaxation with splitting: Error Analysis

$$\mathbf{x}_{k+1} = \mathbf{x}_k + M^{-1}(\mathbf{b} - A\mathbf{x}_k)$$

Jacobi and Gauss-Seidel doesn't converge for all matrices.
Let $\mathbf{x}^* = A^{-1}\mathbf{b}$ be the true solution, and we define the error $\mathbf{e}_{k+1}$ as

$$\begin{aligned}
\mathbf{e}_{k+1} &= \mathbf{x}_{k+1} - \mathbf{x}^* \\
&= \mathbf{x}_k + M^{-1}(\mathbf{b} - A\mathbf{x}_k) - \mathbf{x}^* \\
&= \mathbf{x}_k - \mathbf{x}^* + M^{-1}(A\mathbf{x}^* - A\mathbf{x}_k) \\
&= (I - M^{-1}A)(\mathbf{x}_k - \mathbf{x}^*) \\
&= (I - M^{-1}A)\mathbf{e}_k \\
&= (I - M^{-1}A)^k \mathbf{e}_0
\end{aligned}$$

Relaxation with splitting: Error Analysis

$$\mathbf{x}_{k+1} = \mathbf{x}_k + M^{-1}(\mathbf{b} - A\mathbf{x}_k)$$

Jacobi and Gauss-Seidel doesn't converge for all matrices.
Let $\mathbf{x}^* = A^{-1}\mathbf{b}$ be the true solution, and we define the error $\mathbf{e}_{k+1}$ as

$$\begin{aligned}
\mathbf{e}_{k+1} &= \mathbf{x}_{k+1} - \mathbf{x}^* \\
&= \mathbf{x}_k + M^{-1}(\mathbf{b} - A\mathbf{x}_k) - \mathbf{x}^* \\
&= \mathbf{x}_k - \mathbf{x}^* + M^{-1}(A\mathbf{x}^* - A\mathbf{x}_k) \\
&= (I - M^{-1}A)(\mathbf{x}_k - \mathbf{x}^*) \\
&= (I - M^{-1}A)\mathbf{e}_k \\
&= (I - M^{-1}A)^k \mathbf{e}_0
\end{aligned}$$

Assuming $I - M^{-1}A$ is diagonalizable. For convergence, we need $\lambda_i < 1$ for all eigenvalues λ_i of $I - M^{-1}A$

Relaxation methods are better at correcting oscillatory errors.

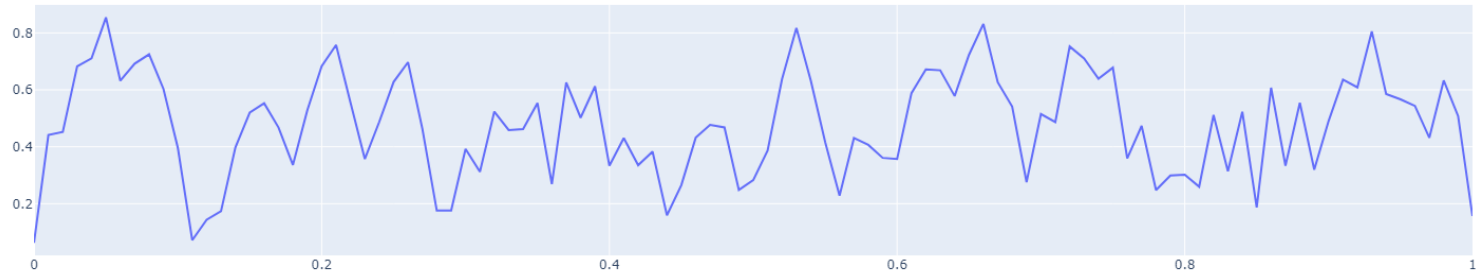


Figure: Fine Grid Iteration 0

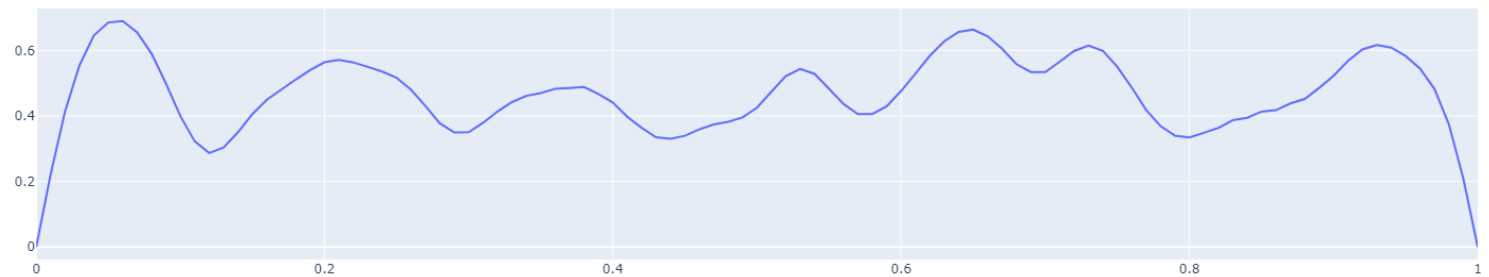


Figure: Fine Grid Iteration 5

Multigrid: the V-cycle

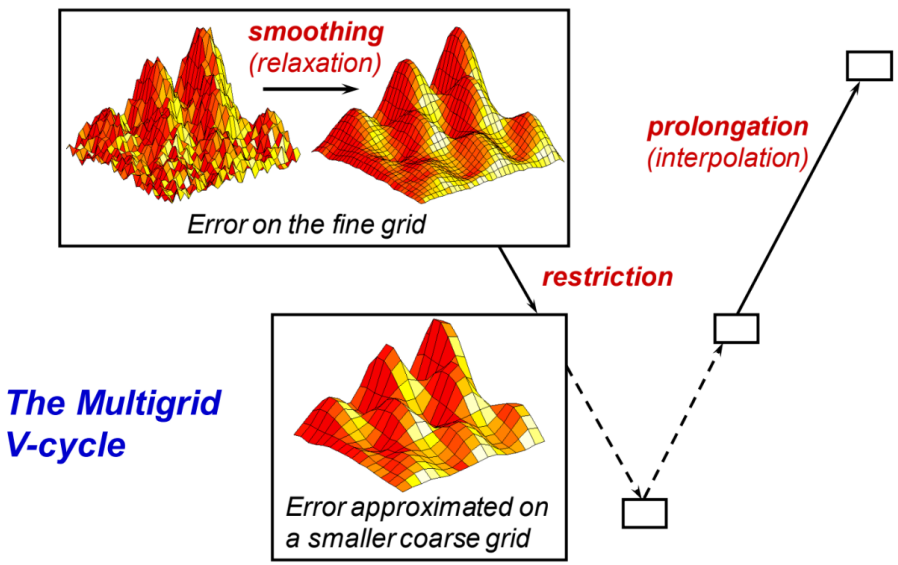


Figure: Multigrid V-cycle

- Multigrid attacks the weakness of relaxation methods by going through a hierarchy of grid sizes.

Multigrid: the V-cycle

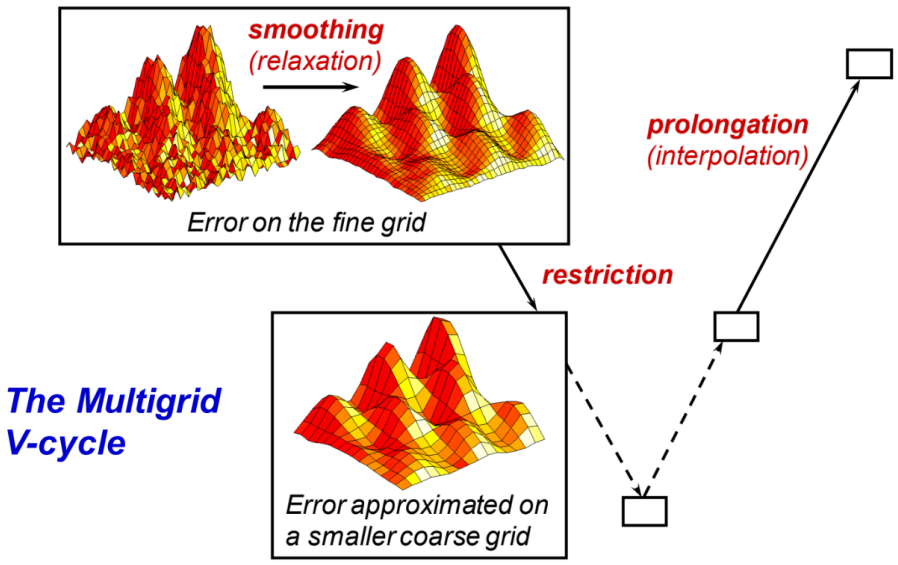


Figure: Multigrid V-cycle

- Multigrid attacks the weakness of relaxation methods by going through a hierarchy of grid sizes.
- Low-frequency (smooth) errors on the fine grid turn into high-frequency (oscillatory) error on the coarser grid and are corrected more effectively through relaxation.

Relaxation methods and Multigrid

Relaxation methods are better at correcting oscillatory errors.

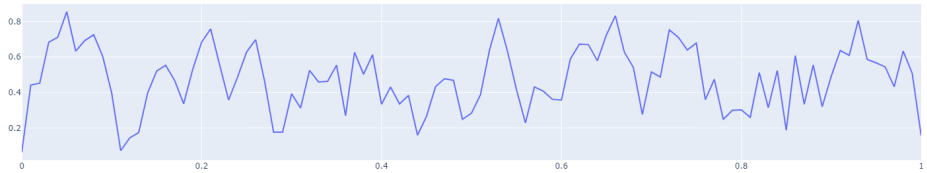


Figure: Fine Grid Iteration 0

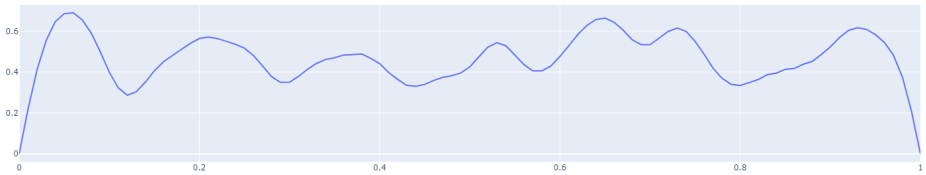


Figure: Fine Grid Iteration 5

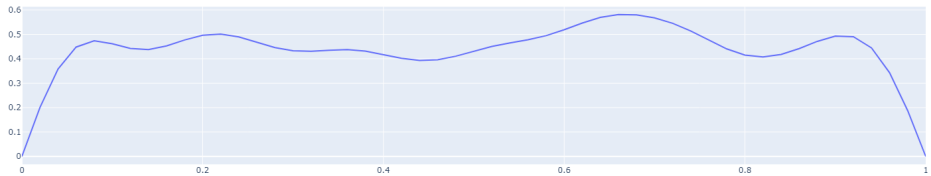


Figure: Coarse Grid Iteration 0

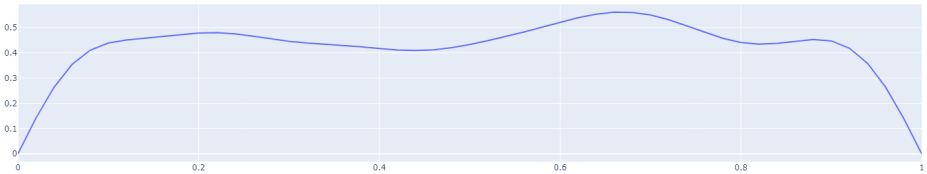


Figure: Coarse Grid Iteration 5

Multigrid reduction in time (MGRIT)

MGRIT algorithm is a parallel-in-time approach for solving time dependent problems that is designed to be non-intrusive.

MGRIT

Consider the diffusion equation $u_t - u_{xx} = f$ with the following discretiztion

$$u_t \approx \frac{u_j^{i+1} - u_j^i}{\delta t}, \quad u_{xx} \approx \frac{u_{j-1}^i - 2u_j^i + u_{j+1}^i}{\delta x^2}$$

where $u_j^i \approx u(x_j, t_i)$ and the update rule

$$\mathbf{u}_i = \Phi \mathbf{u}_{i-1} + \delta t \mathbf{f}_{i-1} \quad \text{for } i = 1, 2 \ldots, n \text{ and } \mathbf{u}_0 = \mathbf{g}_0,$$

where $\mathbf{u}_i \approx \begin{bmatrix} u_0^i & u_1^i & \cdots & u_N^i \end{bmatrix}^T$.

MGRIT

Consider the diffusion equation $u_t - u_{xx} = f$ with the following discretiztion

$$u_t \approx \frac{u_j^{i+1} - u_j^i}{\delta t}, \quad u_{xx} \approx \frac{u_{j-1}^i - 2u_j^i + u_{j+1}^i}{\delta x^2}$$

where $u_j^i \approx u(x_j, t_i)$ and the update rule

$$\mathbf{u}_i = \Phi \mathbf{u}_{i-1} + \delta t \mathbf{f}_{i-1} \quad \text{for } i = 1, 2 \ldots, n \text{ and } \mathbf{u}_0 = \mathbf{g}_0,$$

where $\mathbf{u}_i \approx \begin{bmatrix} u_0^i & u_1^i & \cdots & u_N^i \end{bmatrix}^T$.
Using the partition below, we let $t_i = i\delta t$ for the fine time grid and $T_i = mi\delta t$ for the coarse time grid with coarsening factor m.

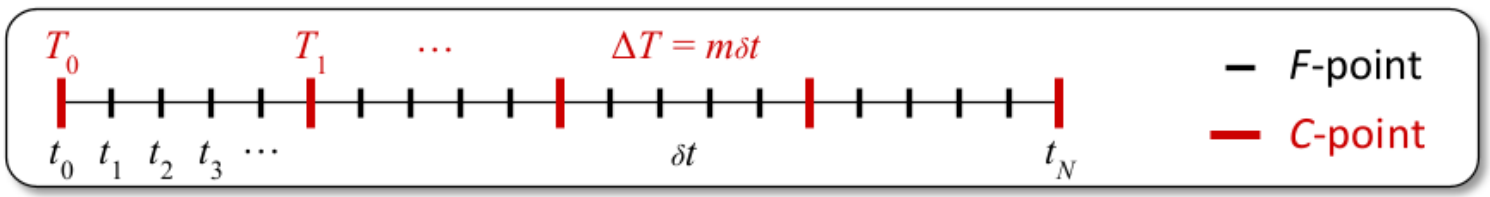


Figure: Coarse and fine time grids

MGRIT: the coarse-grid operator

Then to solve each coarse time step is to solve the following system

$$A_\Delta \mathbf{u} = \begin{pmatrix} I & & & & \\ -\Phi^m & I & & & \\ & -\Phi^m & I & & \\ & & \ddots & \ddots & \\ & & & -\Phi^m & I \end{pmatrix} \begin{pmatrix} u_0 \\ u_m \\ u_{2m} \\ \vdots \\ u_N \end{pmatrix} = \mathbf{f}$$

MGRIT: the coarse-grid operator

Then to solve each coarse time step is to solve the following system

$$A_\Delta \mathbf{u} = \begin{pmatrix} I & & & & \\ -\Phi^m & I & & & \\ & -\Phi^m & I & & \\ & & \ddots & \ddots & \\ & & & -\Phi^m & I \end{pmatrix} \begin{pmatrix} u_0 \\ u_m \\ u_{2m} \\ \vdots \\ u_N \end{pmatrix} = \mathbf{f}$$

However, Φ^m is expensive to compute with, thus we ultimately want to estimate Φ^m with a sparser operator Ψ.

Implementation of basic concepts

- We have implemented basic multigrid and relaxation methods.

Implementation of basic concepts

- We have implemented basic multigrid and relaxation methods.
- We worked on some examples of using neural networks to solve ODEs and matrix approximation.

PyMGRIT and Diffusion Equation

- The code provided in PyMGRIT implemented the heat equation with Backward Euler (implicit).

PyMGRIT and Diffusion Equation

- The code provided in PyMGRIT implemented the heat equation with Backward Euler (implicit).
- By changing the step function, we have worked on the Forward Euler (explicit) version of the equation.

PyMGRIT and Diffusion Equation

- The code provided in PyMGRIT implemented the heat equation with Backward Euler (implicit).
- By changing the step function, we have worked on the Forward Euler (explicit) version of the equation.
- We will integrate machine learning.

Why Machine Learning?

Current approaches of approximating Φ^m applicable to only certain types of PDEs.

Why Machine Learning?

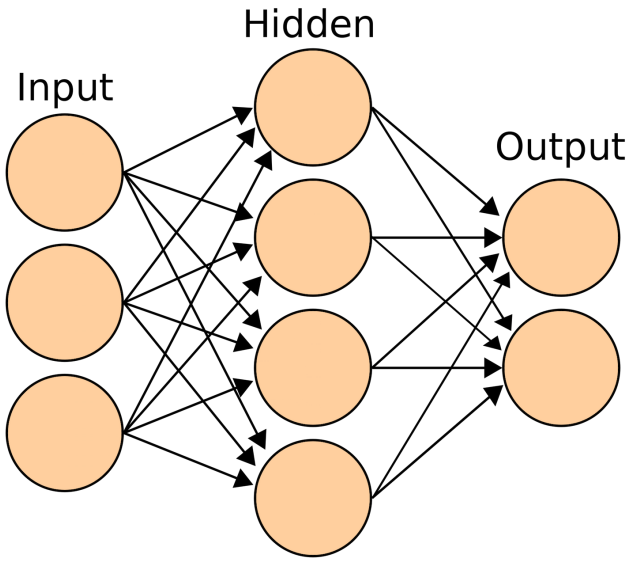
Current approaches of approximating Φ^m applicable to only certain types of PDEs.
We want a "non-intrusive" way to estimate coarse-grid operators.

Why Machine Learning?

Current approaches of approximating Φ^m applicable to only certain types of PDEs.
We want a "non-intrusive" way to estimate coarse-grid operators.
Machine learning is good at coming up at more general solutions.

Current Neural Network Architecture

For 1D diffusion problem:
Input: $(\phi, m) \in \mathbb{R}^3 \times \mathbb{R}$.
Output: $\psi \in \mathbb{R}^3$.



ϕ is the "stencil" of Φ. For example, if

$$\Phi = \begin{bmatrix} -2 & 1 & & & 0 \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ 0 & & & 1 & -2 \end{bmatrix},$$

then $\phi = \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$

Loss Function

- Goal: Find a Ψ that's "spectrally equivalent" to Φ^m. In other words, minimize $\|I - \Psi(\Phi^m)^{-1}\|^2$.

Loss Function

- Goal: Find a Ψ that's "spectrally equivalent" to Φ^m. In other words, minimize $\|I - \Psi(\Phi^m)^{-1}\|^2$.
- Recall, the ℓ^2 norm of a vector $v \in \mathbb{R}^d$ is $\|v\|_{\ell^2}^2 = v_1^2 + \cdots + v_d^2$.

Loss Function

- Goal: Find a Ψ that's "spectrally equivalent" to Φ^m. In other words, minimize $\|I - \Psi(\Phi^m)^{-1}\|^2$.
- Recall, the ℓ^2 norm of a vector $v \in \mathbb{R}^d$ is $\|v\|_{\ell^2}^2 = v_1^2 + \cdots + v_d^2$.
- The norm of a matrix is defined as the induced 2-norm (or spectral norm)

$$\|A\| = \sup_{0 \neq v \in \mathbb{R}^d} \frac{\|Av\|_{\ell^2}}{\|v\|_{\ell^2}}.$$

Loss Function

- Goal: Find a Ψ that's "spectrally equivalent" to Φ^m. In other words, minimize $\|I - \Psi(\Phi^m)^{-1}\|^2$.
- Recall, the ℓ^2 norm of a vector $v \in \mathbb{R}^d$ is $\|v\|_{\ell^2}^2 = v_1^2 + \cdots + v_d^2$.
- The norm of a matrix is defined as the induced 2-norm (or spectral norm)

$$\|A\| = \sup_{0 \neq v \in \mathbb{R}^d} \frac{\|Av\|_{\ell^2}}{\|v\|_{\ell^2}}.$$

- Equivalently, this is the largest magnitude of eigenvalue of A.

Loss Function 1

By choosing some vectors $\{y_k\}_{k=1}^{K}$, we can get the following loss function

Loss Function 1

By choosing some vectors $\{y_k\}_{k=1}^K$, we can get the following loss function

1st Loss function

$$\mathcal{L}_1 = \sum_{k=1}^{K} \frac{\|y_k - (\Psi(\Phi^m)^{-1})y_k\|_{\ell_2}^2}{\|y_k\|_{\ell_2}^2} = \sum_{k=1}^{K} \frac{\|\Phi^m w_k - \Psi w_k\|_{\ell_2}^2}{\|\Phi^m w_k\|_{\ell_2}^2}, \quad y_k = \Phi^m w_k.$$

Loss Function 2

Previous loss function did not work so well in our tests.
Our second idea for a loss function is inspired by Ru Huang et al.

Loss Function 2

Previous loss function did not work so well in our tests.
Our second idea for a loss function is inspired by Ru Huang et al.
Let (λ_k, y_k) be the orthogonal eigenpairs of Φ^m, sorted such that $\lambda_i \geq \lambda_{i+1}$. Let $K \leq d$.

Loss Function 2

Previous loss function did not work so well in our tests.
Our second idea for a loss function is inspired by Ru Huang et al.
Let (λ_k, y_k) be the orthogonal eigenpairs of Φ^m, sorted such that $\lambda_i \geq \lambda_{i+1}$. Let $K \leq d$.
Then we get the following loss function

2nd Loss Function

$$\mathcal{L}_2 = \sum_{k=1}^{K} \|\Phi^m y_k - \Psi y_k\|_{\ell^2}^2.$$

Error analysis

- Consider $\{\lambda_k\}$ be the set of eigenvalues of Φ.

Error analysis

- Consider $\{\lambda_k\}$ be the set of eigenvalues of Φ.
- For diffusion-dominated equations, there are very few $|\lambda_k| \approx 1$ and many $|\lambda_k| \ll 1$.

Error analysis

- Consider $\{\lambda_k\}$ be the set of eigenvalues of Φ.
- For diffusion-dominated equations, there are very few $|\lambda_k| \approx 1$ and many $|\lambda_k| \ll 1$.
- For example, for the equation $u_t = u_{xx}$, if we consider N gridpoints, then the eigenvalues are given by

$$\lambda_k = -2 + 2\cos\left(\frac{k\pi}{N+1}\right), 1 \leq k \leq N.$$

Error analysis

- Consider $\{\lambda_k\}$ be the set of eigenvalues of Φ.
- For diffusion-dominated equations, there are very few $|\lambda_k| \approx 1$ and many $|\lambda_k| \ll 1$.
- For example, for the equation $u_t = u_{xx}$, if we consider N gridpoints, then the eigenvalues are given by

$$\lambda_k = -2 + 2\cos\left(\frac{k\pi}{N+1}\right), 1 \leq k \leq N.$$

- This makes approximation for Ψ easier as the rate of convergence is faster.

Error analysis

- Consider $\{\lambda_k\}$ be the set of eigenvalues of Φ.
- For diffusion-dominated equations, there are very few $|\lambda_k| \approx 1$ and many $|\lambda_k| \ll 1$.
- For example, for the equation $u_t = u_{xx}$, if we consider N gridpoints, then the eigenvalues are given by

$$\lambda_k = -2 + 2\cos\left(\frac{k\pi}{N+1}\right), 1 \leq k \leq N.$$

- This makes approximation for Ψ easier as the rate of convergence is faster.
- However, the distribution of $|\lambda_k|$ varies between different types of PDEs (hyperbolic, elliptic, etc).

Efficiency Results

- By using the first loss function

$$\mathcal{L}_1 = \sum_{k=1}^{K} \frac{\|\Phi^m w_k - \Psi w_k\|_{\ell_2}^2}{\|\Phi^m w_k\|_{\ell_2}^2}$$

and 32 random unit vectors $w_1, \ldots, w_{32}$, with 256 epochs, we have the loss be approximately 10^{-1}.

Efficiency Results

- By using the first loss function

$$\mathcal{L}_1 = \sum_{k=1}^{K} \frac{\|\Phi^m w_k - \Psi w_k\|_{\ell_2}^2}{\|\Phi^m w_k\|_{\ell_2}^2}$$

  and 32 random unit vectors $w_1, \ldots, w_{32}$, with 256 epochs, we have the loss be approximately 10^{-1}.
- However, it seems that the stencil returned is not close enough.

Efficiency Results

- By using the first loss function

$$\mathcal{L}_1 = \sum_{k=1}^{K} \frac{\|\Phi^m w_k - \Psi w_k\|_{\ell_2}^2}{\|\Phi^m w_k\|_{\ell_2}^2}$$

  and 32 random unit vectors $w_1, \ldots, w_{32}$, with 256 epochs, we have the loss be approximately 10^{-1}.
- However, it seems that the stencil returned is not close enough.
- The second loss function

$$\mathcal{L}_2 = \sum_{k=1}^{K} \|\Phi^m y_k - \Psi y_k\|_{\ell^2}^2$$

  seems to align more with the testing stencil.

Next steps

- 1D Diffusion only proof-of-concept – generalize to more types of PDEs.
- Currently, Ψ hard-coded to be tridiagonal. Use second neural network to select nonzero entries of Ψ, following Ru Huang et als' paper.
- Experiment with loss function: perhaps weight different eigenvectors based on λ.
- Use bootstrapping ideas.
  - Use the output to train the model.
  - Solve $A\mathbf{u} = \mathbf{0}$ to generate error vectors and update the training set with them.
  - We want convergence rate to be faster than linear so we work on the null-space so bootstrap works on those error components.

Acknowledgement

- Thank you for listening!
- Questions?