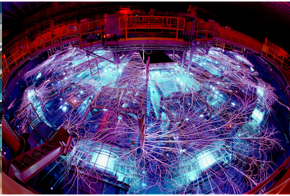
Exceptional service in the national interest

Sandia National Laboratories



Reduced Representation and Compression Techniques for Patch-Based Relaxation

Graham Harper*, Ray Tuminaro
Center for Computing Research
Sandia National Laboratories

ICIAM 2023/8/25

Sandia National Laboratories is a multimission laboratory managed and opera
owned subsidiary of Honeywell International Inc., for the U.S. Department of E
DE-NA0003525.

Acknowledgements

- Trilinos/MueLu/Ifpack2
- Funding: Harper's LDRD, Ridzal's LDRD, Tuminaro's ASCR
- Team: Ray Tuminaro
- Sandia National Laboratories LDRD Office, U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research

Outline

Background

Smoothers & Patch Smoothers

Details

Observations

Main Idea

Algorithms

Results

Timings

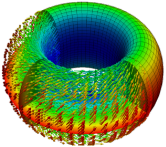
Multigrid

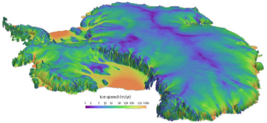
Visualization

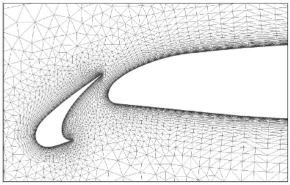
Conclusion

Overview for Smoothers

- Traditional simple smoothers (Jacobi) work well for low-order problems, but tend to struggle as the **polynomial degree** increases or **coupling** increases.

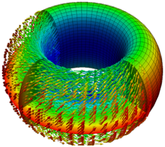


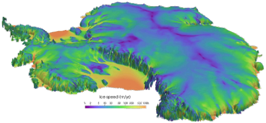


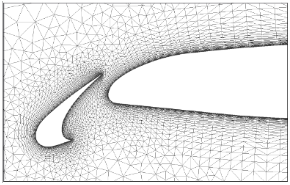


Overview for Smoothers

- Traditional simple smoothers (Jacobi) work well for low-order problems, but tend to struggle as the **polynomial degree** increases or **coupling** increases.



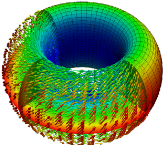
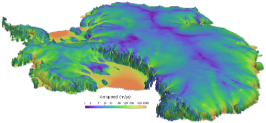
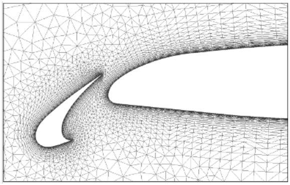




- Patch-based smoothers reduce iteration counts at the cost of storing/solving many small linear systems.

Overview for Smoothers

- Traditional simple smoothers (Jacobi) work well for low-order problems, but tend to struggle as the **polynomial degree** increases or **coupling** increases.

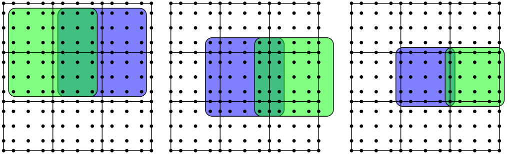
- Patch-based smoothers reduce iteration counts at the cost of storing/solving many small linear systems.
- What if we could reduce those costs?

Patch-Based Smoothers

- Recall a smoother takes the form

$$\mathbf{x} \leftarrow \mathbf{x} + \omega \tilde{M}^{-1}(\mathbf{b} - A\mathbf{x}) \tag{1}$$

- Consider an (overlapping) domain decomposition with n_p domains and boolean restriction operators R_i.

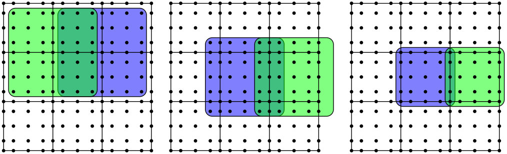


Patch-Based Smoothers

- Recall a smoother takes the form

$$\mathbf{x} \leftarrow \mathbf{x} + \omega \tilde{M}^{-1}(\mathbf{b} - A\mathbf{x}) \quad (1)$$

- Consider an (overlapping) domain decomposition with n_p domains and boolean restriction operators R_i.



- A patch-based smoother utilizes

$$\tilde{M}^{-1} = \sum_{i=1}^{n_p} R_i^T W_i A_i^{-1} R_i \quad (2)$$

Patch-Based Smoothers

$$\tilde{M}^{-1} = \sum_{i=1}^{n_p} R_i^T W_i A_i^{-1} R_i$$

- R_i is the boolean restriction
- W_i are the global weights, $(\text{overlap})^{-1}$
- $A_i = R_i A R_i^T$ is the $p_s \times p_s$ patch matrix

Patch-Based Smoothers

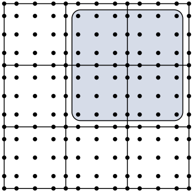
$$\tilde{M}^{-1} = \sum_{i=1}^{n_p} R_i^T W_i A_i^{-1} R_i$$

- R_i is the boolean restriction
- W_i are the global weights, $(\text{overlap})^{-1}$
- $A_i = R_i A R_i^T$ is the $p_s \times p_s$ patch matrix

Patch methods:

- Pros: Converge more quickly, handle complex coupling more effectively
- Cons: More expensive to compute, similar storage cost to original matrix

Clarifications

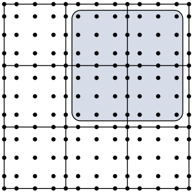


(a) Vertex-star patches

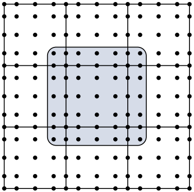
There are many kinds of patch smoothers:

(a) Scales for high values of p, difficult solves, depends on connectivity

Clarifications



(a) Vertex-star patches



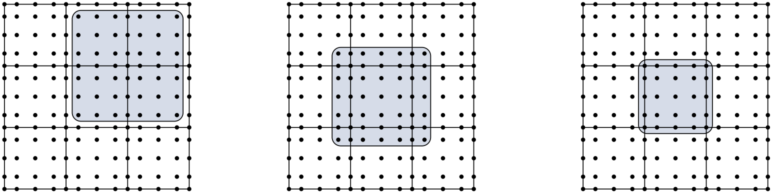
(b) Cell-centered patches

There are many kinds of patch smoothers:

(a) Scales for high values of p, difficult solves, depends on connectivity

(b) Allows for structured patches, not p-robust

Clarifications



(a) Vertex-star patches (b) Cell-centered patches (c) Cell-restricted patches

There are many kinds of patch smoothers:

(a) Scales for high values of p, difficult solves, depends on connectivity

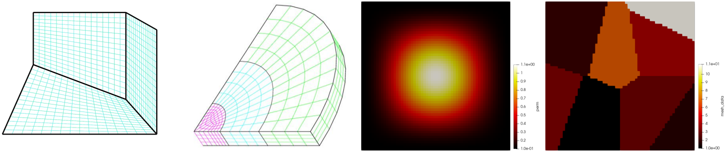
(b) Allows for structured patches, not p-robust

(c) Most efficient to detect, most structure to exploit

*left figures: P. Brubeck, P. Farrell, *A Scalable and Robust Vertex-Star Relaxation for High-Order FEM*. SISC 2022

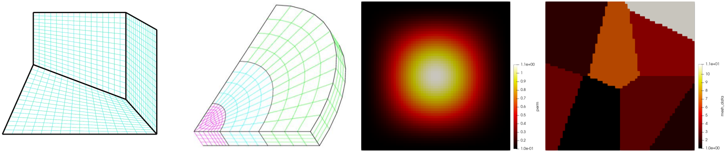
Observations

1. In many applications of interest, if you zoom in far enough, patterns appear.



Observations

1. In many applications of interest, if you zoom in far enough, patterns appear.

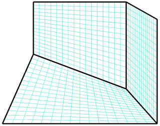
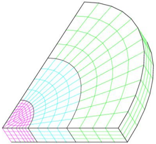
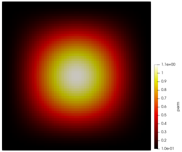
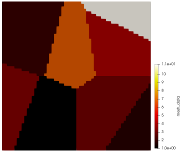


2. If patches are “similar enough,” why store and solve them multiple times?

Observations

Sandia National Laboratories

1. In many applications of interest, if you zoom in far enough, patterns appear.

2. If patches are "similar enough," why store and solve them multiple times?

Question: How "bad" can the approximation get before the solver (smoother-only or multigrid) starts to struggle?

The Main Idea

- Construct a database of patches $\mathcal{B} = \{B_1, B_2, \ldots, B_{m_p}\}$.
- Criterion 1: $\mathcal{B}$ should approximate $\{A_i\}$ sufficiently well.
- Criterion 2: $\mathcal{B}$ should be small compared to $\{A_i\}$.

The Main Idea

- Construct a database of patches $\mathcal{B} = \{B_1, B_2, \ldots, B_{m_p}\}$.
- Criterion 1: $\mathcal{B}$ should approximate $\{A_i\}$ sufficiently well.
- Criterion 2: $\mathcal{B}$ should be small compared to $\{A_i\}$.

Seek $\mathcal{B} = \{B_1, B_2, \ldots, B_{m_p}\}$ and mapping $\phi : \{1, \ldots, n_p\} \to \{1, \ldots, m_p\}$ minimizing

$$\mathcal{L}(\mathcal{B}, \phi) = \beta|\mathcal{B}| + \sum_{k=1}^{n_p} \|I - A_k B_{\phi(k)}^{-1}\|_2^2, \qquad (3)$$

balancing efficiency and accuracy.

Also referred to as a **sparse approximation problem**.

Greedy Algorithm

Algorithm 1 Greedy Construction of $\mathcal{B}$, $\phi()$

Input: $\{A_1, A_2, \ldots, A_{n_p}\}$, ε
$\mathcal{B} := \{\}$, $\vec{\phi} = 0$
for $i = 1, \ldots, n_p$ **do**
    match:=false;
    for $j = 1, \ldots, m_p$ **do**
        if $\|I - A_i B_j^{-1}\|_2 < \varepsilon$ **then**
            match=true, $\phi(i) = j$, break;
        end if
    end for
    if match==false **then**
        append A_i to $\mathcal{B}$, $\phi(i) = |\mathcal{B}|$;
    end if
end for
Output: $\mathcal{B} = \{B_1, B_2, \ldots, B_{m_p}\}$, $\vec{\phi}$

← check patch against database

← if no matches, store patch explicitly

Greedy Local Error Bounds

Bonus: $\|I - A_i B_{\phi(i)}^{-1}\| < \varepsilon$, ensures solver quality. If

$$A_i x = v, \tag{4}$$

then approximating $B_{\phi(i)} y = v$ yields

$$\begin{aligned} \|x - y\| &= \|A_i^{-1} v - B_{\phi(i)}^{-1} v\| \\ &= \|A_i^{-1}(I - A_i B_{\phi(i)}^{-1}) v\| \\ &< \varepsilon \|A_i^{-1}\| \|v\|. \end{aligned} \tag{5}$$

*then extend to the global problem

Clustering Algorithm

Algorithm 3 Clustering Construction of $\mathcal{B}$, $\phi()$

Input: $\{A_1, A_2, \ldots, A_{n_p}\}$, m_p ← choose number of clusters
$\vec{\phi} = \text{randperm}(n_p, m_p)$;
$B_i = A_{\phi(i)}$, $i = 1, \ldots, m_p$;
while not converged **do**
  for $i = 1, \ldots, n_p$ **do**
    for $j = 1, \ldots, m_p$ **do**
      $d_{ij} = d(A_i, B_j)$; ← $d(A_i, B_j) = \|I - A_i B_j^{-1}\|_2$
    end for
    $\phi(i) = \text{argmin}_j(d_{ij})$;
  end for
  $B_i = \frac{1}{n_i} \sum_{j \text{ in cluster } i} A_j$, $i = 1, \ldots, m_p$; *treat boundaries separately!
end while
Output: $\mathcal{B} = \{B_1, B_2, \ldots, B_{m_p}\}$, $\vec{\phi}$

Clustering Local Error Bounds

For clustering, the idea is similar. However, $d(A_i, B_{\phi(i)}) := \|(I - A_i B_{\phi(i)}^{-1}\|$ is not bounded a priori.

$$\begin{aligned} \|x - y\| &= \|A_i^{-1} v - B_{\phi(i)}^{-1} v\| \\ &= \|A_i^{-1}(I - A_i B_{\phi(i)}^{-1}) v\| \\ &< \max_i d(A_i, B_{\phi(i)}) \|A_i^{-1}\| \|v\|. \end{aligned} \tag{6}$$

may be bounded by cluster diameters.

Clustering Variations

There are a number of different ways to approach clustering.

1. **Entrywise** k**-means** $d(A_i, B_j) = \|A_i - B_j\|_{\ell_1}$,
   B_j^{-1} = inverse of entrywise cluster average.
2. **Spectral** k**-means** $d(A_i, B_j) = \|I - A_i B_j^{-1}\|_2$,
   B_j^{-1} = inverse of entrywise cluster average.
3. **Variance-minimizing clustering** $d(A_i, B_j) = \|I - A_i B_j^{-1}\|_2$,
   B_j^{-1} = inverse of member minimizing in-cluster variance.

Bootstrapped Algorithm

"But clustering is slow!"

Make it faster by bootstrapping:

1. Run Algorithm 1, obtain $\mathcal{B}$
2. Initialize Algorithm 2 clustering with $\mathcal{B}$
3. Do very few iterations of Algorithm 2

Pros:

- Better database assignments than Algorithm 1
- Faster time to result than Algorithm 2

Cons:

- More complex
- Slower than Algorithm 1

Poisson Equation

$$-\nabla \cdot (\rho(x, y)\nabla u) = f. \tag{7}$$

Dirichlet BCs, $p = 2, 3, 4, 5$ FEMs, uniform grid, solution

$$u = \sin(\pi x)\sin(\pi y)$$

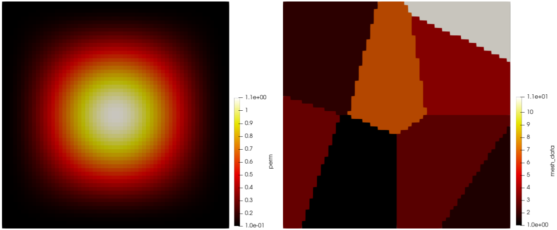


Figure: (left) smooth coefficient, (right) piecewise discontinuous coefficient.
Not pictured: constant coefficient

Timing Results

Configuration: $\rho = 1$ Poisson, $20 \times 20 \times 20$ grid, $p = 5$
GMRES with 10^{-7} tolerance, $\varepsilon = 10^{-7}$, $|\mathcal{B}| = 8000, 27$.
Implemented in **Trilinos/Ifpack2** as a preconditioner using LAPACK GETRF, GETRS. Used as additive Schwarz inner solve. Compress if $\|A_i - B_j\|_{\ell_1} < \varepsilon$. 39 iterations.

Configuration ($N = 5$) runs	Setup (s) mean$\pm$std	Apply (s) mean$\pm$std	Storage
No compression	269.52 ± 1.46	39.10 ± 0.46	2.8GB
Compression	253.12 ± 0.61	28.87 ± 0.047	9.6MB $+$ 62.5KB

Single node of Attaway supercomputer, Intel Xeon Gold 6140 Processor. Cache size is 24.75MB.

Timing Results

Configuration: $\rho = 1$ Poisson, $20 \times 20 \times 20$ grid, $p = 5$
GMRES with 10^{-7} tolerance, $\varepsilon = 10^{-7}$, $|\mathcal{B}| = 8000, 27$.

Implemented in **Trilinos/Ifpack2** as a preconditioner using LAPACK GETRF, GETRS. Used as additive Schwarz inner solve. Compress if $\|A_i - B_j\|_{\ell_1} < \varepsilon$. 39 iterations.

Configuration ($N = 5$) runs	Setup (s) mean±std	Apply (s) mean±std	Storage
No compression	269.52 ± 1.46	39.10 ± 0.46	2.8GB
Compression	253.12 ± 0.61	28.87 ± 0.047	9.6MB + 62.5KB

Single node of Attaway supercomputer, Intel Xeon Gold 6140 Processor. Cache size is 24.75MB.

Jacobi would cost 7.9MB.

Burgers' Equation

$$\frac{du}{dt} + \nabla \cdot \left(\frac{1}{2}\vec{\nu}u^2 - g(u)\nabla u \right) = f, \qquad (8)$$

where $\vec{\nu} = [1,1]^T$, g is nonlinear entropy-viscosity term, and SUPG stabilization is utilized in the discretization. 100×100 grid.

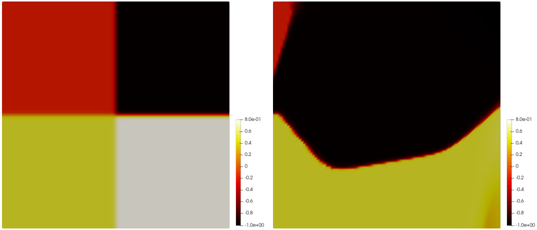
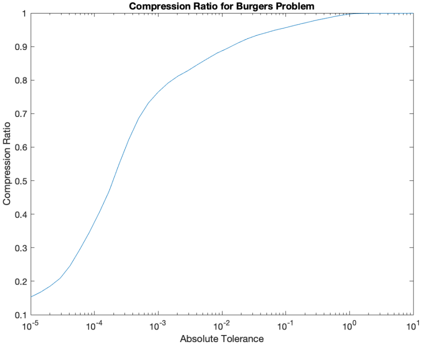


Figure: (left) $t = 0$ starting profile for Burgers' equation (right) $t = 1$ end profile for Burgers' equation

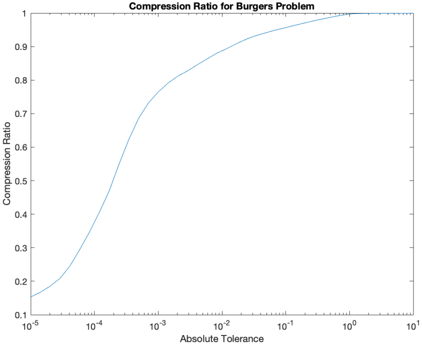
Burgers' Equation

Plotting greedy algorithm $\frac{n_p - |B|}{n_p}$ against ε shows it's compressible.



Burgers' Equation

Plotting greedy algorithm $\frac{n_p - |B|}{n_p}$ against ε shows it's compressible.

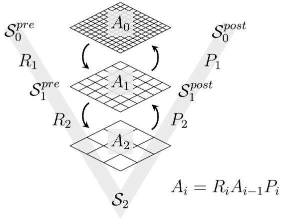


Algorithm\Database Size	10000	371	293	213	146	76	25	13
Greedy Tolerance 1	224	240	310	238	278	463	829	882
Entrywise k-means	224	342	343	377	317	353	416	420

Poisson Multigrid Method

We use the following multigrid configuration for Poisson:

- $\omega = 0.5$,
- $(\nu_1, \nu_2) = (1, 0)$,
- $N = 2$ V-cycle levels,
- Case 1: P uniform linear interp.
  Case 2: P from Ruge-Stüben AMG
- $R = P^T$, $A_c = RAP$



With the data-driven smoother, compute $\mathcal{B}, \vec{\phi}$ during setup; use them during apply.

Results (Smooth Permeability, Linear P)

	Algorithm\Database Size	3600	74	35	18	15	13	7	6
$p=2$	Greedy Tolerance 1	11	12	13	14	17	26	61	68
	Spectral k-means 3	11	12	12	14	16	17	–	–
	Var-Minimizing Clustering	11	12	12	13	14	15	–	–
	Entrywise k-means	11	11	12	12	14	16	–	–
$p=3$	Algorithm\Database Size	3600	71	34	18	15	13	8	6
	Greedy Tolerance 1	12	13	13	15	18	26	58	69
	Spectral k-means 3	12	12	12	15	17	18	–	–
	Var-Minimizing Clustering	12	12	12	13	14	15	–	–
	Entrywise k-means	12	12	12	12	14	16	–	–
$p=4$	Algorithm\Database Size	3600	71	34	18	15	13	8	7
	Greedy Tolerance 1	14	14	15	17	20	29	68	67
	Spectral k-means 3	14	14	14	17	19	20	–	–
	Var-Minimizing Clustering	14	14	14	15	16	17	–	–
	Entrywise k-means	14	14	14	14	16	18	–	–
$p=5$	Algorithm\Database Size	3600	73	35	18	15	13	9	7
	Greedy Tolerance 1	15	16	17	19	22	32	78	76
	Spectral k-means 3	15	16	16	19	21	22	–	–
	Var-Minimizing Clustering	15	15	16	17	18	19	–	–
	Entrywise k-means	15	15	15	16	18	20	–	–

Results (Piecewise Constant Permeability, Linear P)

	Algorithm\Database Size	3600	131	113	96	52	25	5	3
$p=2$	Greedy Tolerance 1	11	12	12	14	17	22	47	52
	Spectral k-means 3	11	17	19	19	26	27	–	–
	Var-Minimizing Clustering	11	20	20	20	29	31	–	–
	Entrywise k-means	11	12	17	17	26	31	–	–
	Algorithm\Database Size	3600	131	114	100	59	32	8	5
$p=3$	Greedy Tolerance 1	12	12	13	14	17	26	48	50
	Spectral k-means 3	12	20	26	26	34	35	–	–
	Var-Minimizing Clustering	12	24	24	24	32	32	–	–
	Entrywise k-means	12	13	19	19	29	33	–	–
	Algorithm\Database Size	3600	130	114	103	70	36	12	9
$p=4$	Greedy Tolerance 1	14	14	15	15	19	28	51	53
	Spectral k-means 3	14	23	32	32	33	38	36	–
	Var-Minimizing Clustering	14	27	27	27	27	38	37	–
	Entrywise k-means	14	16	22	22	22	35	37	–
	Algorithm\Database Size	3600	130	116	103	78	38	15	10
$p=5$	Greedy Tolerance 1	15	16	16	17	21	29	60	64
	Spectral k-means 3	15	25	35	30	35	40	43	39
	Var-Minimizing Clustering	15	28	28	54	47	40	40	44
	Bootstrapped Var-Minimizing	15	16	16	17	22	26	44	39
	Entrywise k-means	15	18	25	25	25	38	40	38

Maxwell's Equations, Distorted Mesh

Sandia National Laboratories

$$-\nabla \times \nabla \times \mathbf{u} - \sigma \mathbf{u} = \mathbf{f}. \tag{9}$$

where $\sigma = 10^{-4}$. Arnold-Falk-Winther patches are utilized. $30 \times 30 \times 30$ grid, $29^3 = 24389$ patches. Perturbed structured grid

$$\begin{aligned} x &= \hat{x} + 0.6\sin(\pi\hat{x}/2)\sin(\pi\hat{y}/2)\sin(\pi\hat{z}/2), \\ y &= \hat{y} + 0.6\cos(\pi\hat{x}/2)\cos(\pi\hat{y}/2)\cos(\pi\hat{z}/2), \\ z &= \hat{z} + 0.6\cos(\pi\hat{x}/2)\cos(\pi\hat{y}/2). \end{aligned} \tag{10}$$

Solution

$$\mathbf{u} = \begin{bmatrix} \sin(\pi x)\sin(\pi y) \\ y \\ z \end{bmatrix}.$$

Results (Maxwell's Equations, Distorted Mesh)

Unrestarted GMRES, relative tolerance 10^{-6}, initial guess $10^2 x + 10y + z$.
2-level MG, two sweeps as a pre-smoother with a damping parameter $\omega = 0.7$.

Case 1: linear interp. P (ignoring distortion)

Algorithm\Database Size	24389	361	129	43	16
Greedy Tolerance 1	33	34	34	35	DNC
Spectral k-means 3	33	33	DNC	DNC	DNC

Case 2: Ruge-Stüben AMG

Algorithm\Database Size	24389	361	129	43	16
Greedy Tolerance 1	56	57	58	60	DNC
Spectral k-means 3	56	57	DNC	DNC	DNC

Results (Database Mappings)

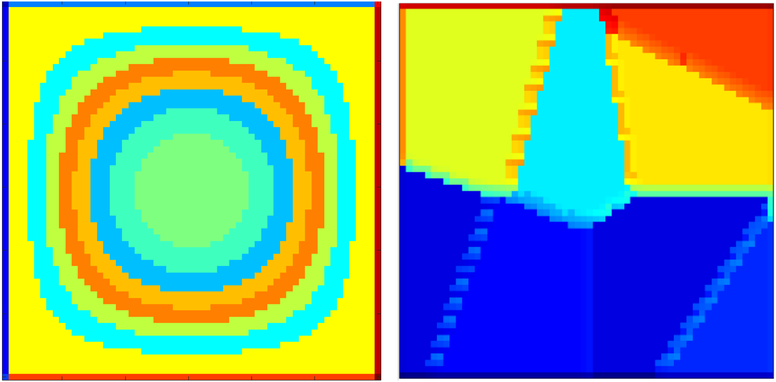


Figure: Visualization of database ϕ mapping. (left) clustering algorithm on smooth permeability, (right) greedy algorithm on piecewise constant permeability

Results (Other Visualization)

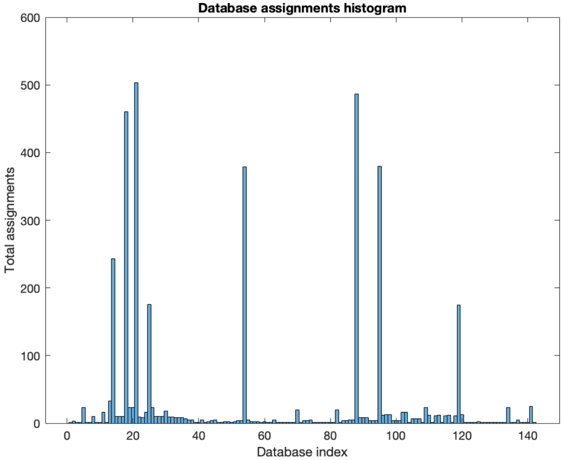


Figure: Database assignments for piecewise discontinuous problem, $\varepsilon = 10^{-7}$

Conclusion

Takeaways:

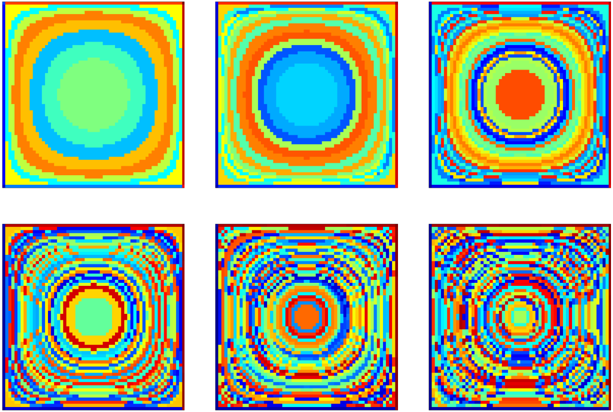
- Memory cost $= \mathcal{O}(\text{Jacobi})$
- Faster apply than without database
- Generalizes for any structure detection method

Current/Future Work:

- Utilize Kokkos, not LAPACK
- Time-dependent problems
- Ridzal's LDRD: R-adaptivity to detect and enhance compressibility
- Unsupervised deep learning

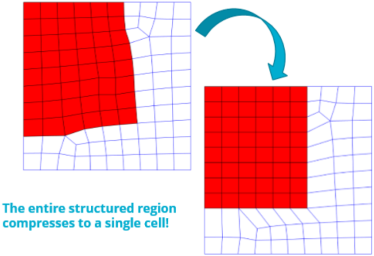
Thank You

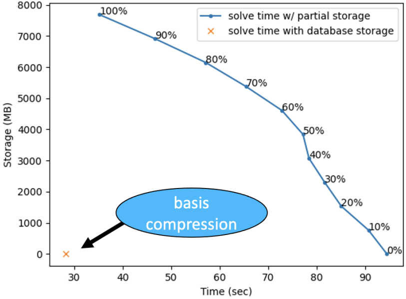
- Questions?



Supplemental Slides

R-adaptivity refers to moving mesh nodes.
We're interested in the following scenario:





for finite element basis reuse. This also results in patch reuse.

Supplemental Slides

Preliminary investigations by summer students Nichole Etienne (Emory) and Eugene Agyei-Kodie (Michigan State) on different clustering approaches