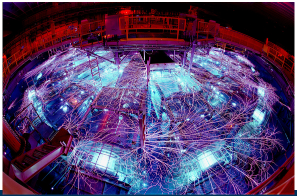
*Exceptional service in the national interest*



# Krylov Solvers and Preconditioning

Jonathan Hu and Christian Glusa

ATPESC 2019
Q Center, St. Charles, IL (USA)
August 6, 2019

Discretization of partial differential equations gives rise to large linear systems of equations

$$\boldsymbol{A}\vec{x} = \vec{b},$$

where $\boldsymbol{A}$ is sparse, i.e. only a few non-zero entries per row.

## Example

2D Poisson equation:

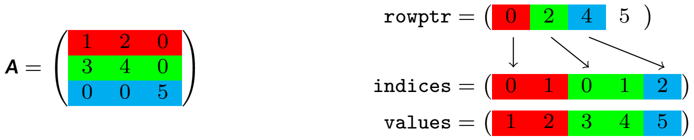
$$\begin{aligned} -\Delta u &= f \text{ in } \Omega = [0,1]^2, \\ u &= 0 \text{ on } \partial\Omega. \end{aligned}$$

Central finite differences on a uniform mesh $\{x_{i,j}\}$:

$$\begin{aligned} 4u_{i,j} - u_{i,j+1} - u_{i,j-1} - u_{i+1,j} - u_{i-1,j} &= f(x_{i,j})\Delta x \quad && \text{if } x_{i,j} \notin \partial\Omega, \\ u_{i,j} &= 0 && \text{if } x_{i,j} \in \partial\Omega. \end{aligned}$$

$\rightarrow$ 5 entries or less per row of $\boldsymbol{A}$.

Instead of dense format, keep matrix $\boldsymbol{A}$ in a sparse format e.g. *compressed sparse row* (CSR):



$$\boldsymbol{A} = \begin{pmatrix} 1 & 2 & 0 \\ 3 & 4 & 0 \\ 0 & 0 & 5 \end{pmatrix}$$

$$\begin{aligned} \texttt{rowptr} &= \begin{pmatrix} 0 & 2 & 4 & 5 \end{pmatrix} \\ \texttt{indices} &= \begin{pmatrix} 0 & 1 & 0 & 1 & 2 \end{pmatrix} \\ \texttt{values} &= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \end{pmatrix} \end{aligned}$$

## Available solvers

Solve

$$\boldsymbol{A}\vec{x} = \vec{b}.$$

**Option 1:** Direct solvers (think Gaussian elimination)

- Factorisation scales as $\mathcal{O}(n^3)$.
- Factors are a lot denser than $\boldsymbol{A}$ $\rightarrow$ memory cost.
- Parallel implementation not straightforward.
- Does not require a lot of information about the structure of $\boldsymbol{A}$.

**Observation**

$\boldsymbol{A}$ has $\mathcal{O}(n)$ non-zero entries. $\rightarrow$ Optimal complexity for a solve is $\mathcal{O}(n)$ operations.

**Option 2:** Iterative solvers

- Exploit an operation that has $\mathcal{O}(n)$ complexity: mat-vec.
- Easy to parallelize.
- Can have small memory footprint. (In the best case, we only need to keep a single vector.)
- More restrictions on required structure of $\boldsymbol{A}$.

## Available solvers

Solve

$$\mathbf{A}\vec{x} = \vec{b}.$$

**Option 1:** Direct solvers (think Gaussian elimination)

- Factorisation scales as $\mathcal{O}(n^3)$.
- Factors are a lot denser than $\mathbf{A}$ $\rightarrow$ memory cost.
- Parallel implementation not straightforward.
- Does not require a lot of information about the structure of $\mathbf{A}$.

**Observation**

$\mathbf{A}$ has $\mathcal{O}(n)$ non-zero entries. $\rightarrow$ Optimal complexity for a solve is $\mathcal{O}(n)$ operations.

**Option 2:** Iterative solvers

- Exploit an operation that has $\mathcal{O}(n)$ complexity: mat-vec.
- Easy to parallelize.
- Can have small memory footprint. (In the best case, we only need to keep a single vector.)
- More restrictions on required structure of $\mathbf{A}$.

## Krylov methods

Based on mat-vecs, we can compute

$$\vec{y}^0 = \vec{x^0} \qquad \text{(“initial guess”’)}$$

$$\vec{y}^{k+1} = \vec{y}^k + \underbrace{\left(\vec{b} - \boldsymbol{A}\vec{y}^k\right)}_{\text{“residual”}}$$

and recombine in some smart way to obtain an approximate solution

$$\vec{x}^K = \sum_{k=0}^{K} \alpha_k \vec{y}^k.$$

The values of $\alpha_k$ typically involve inner products between vectors in the so-called *Krylov space*
span $\left\{\vec{y}^k\right\} = \left\{\vec{x^0}, \boldsymbol{A}\vec{x}^0, \boldsymbol{A}^2\vec{x}^0, \boldsymbol{A}^3\vec{x}^0, \ldots\right\}$.

- Keeping the entire Krylov space can be quite expensive.
- Computing inner products involves an all-reduce which can be costly at large scale.

Two particular Krylov methods:

- Conjugate gradient (CG)
  - Use a short recurrence, i.e. does not keep the whole Krylov space around.
  - Provably works for symmetric positive definite (spd) $\boldsymbol{A}$.
- Generalized Minimum Residual (GMRES, GMRES($K$))
  - Works for unsymmetric systems.
  - GMRES keeps the whole Krylov space around.
  - GMRES($K$) discards the Krylov space after $K$ iterations.

## Convergence of Krylov methods

The following holds for CG:

$$\left\|\vec{x}^K - \vec{x}\right\| \leq \left(1 - 1/\sqrt{\kappa(\boldsymbol{A})}\right)^K \left\|\vec{x}^0 - \vec{x}\right\|,$$

where $\kappa(\boldsymbol{A})$ is the condition number of $\boldsymbol{A}$:

$$\kappa(\boldsymbol{A}) = \|\boldsymbol{A}\| \left\|\boldsymbol{A}^{-1}\right\|.$$

It turns out that this is a common theme with Krylov methods.
The condition number can be seen as a measure of how hard it is to solve the system.

**Idea**

Reduce the condition number. ("*Preconditioning*")

Instead of solving

$$\boldsymbol{A}\vec{x} = \vec{b},$$

solve

$$\boldsymbol{P}\boldsymbol{A}\vec{x} = \boldsymbol{P}\vec{b} \qquad \text{or} \qquad \boldsymbol{A}\boldsymbol{P}\vec{z} = \vec{b}, \quad \vec{x} = \boldsymbol{P}\vec{z}$$

with preconditioner $\boldsymbol{P}$ so that $\kappa(\boldsymbol{P}\boldsymbol{A}) \ll \kappa(\boldsymbol{A})$.
Two conflicting requirements:

- Multiplication with $\boldsymbol{P}$ should be comparable in cost to $\boldsymbol{A}$.
- $\boldsymbol{P} \approx \boldsymbol{A}^{-1}$.

## Some simple preconditioners

- Jacobi: $\boldsymbol{P} = \boldsymbol{D}^{-1}$, where $\boldsymbol{D}$ is the diagonal of $\boldsymbol{A}$.
- Gauss-Seidel: $\boldsymbol{P} = (\boldsymbol{D} + \boldsymbol{L})^{-1}$, where $\boldsymbol{L}$ is the lower or upper triangular part of $\boldsymbol{A}$.
- Polynomial preconditioners: $\boldsymbol{P} = p(\boldsymbol{A})$, where $p$ is some carefully chosen polynomial.
- Incomplete factorizations such as ILU or Incomplete Cholesky.

## The Trilinos project

- Collection of interoperable packages for the solution of large-scale, complex multiphysics engineering and scientific problems
- discretization in space & time, mesh and graph tools, automatic differentiation, linear & nonlinear solvers & preconditioners, eigen-solvers, optimization, UQ, ...
- (Mostly) C++ and object-oriented
- Support for hybrid (MPI+X) parallelism, $X \in \{\text{OpenMP, CUDA, Pthreads}, \ldots\}$
- Open source, primarily developed at Sandia

**Belos - iterative linear solvers**

- Standard methods:
  - Conjugate Gradients (CG), Generalized Minimal Residual (GMRES)
  - TFQMR, BiCGStab, MINRES, Richardson / fixed-point
- Advanced methods:
  - Block GMRES, block CG/BiCG
  - Hybrid GMRES, CGRODR (block recycling GMRES)
  - TSQR (tall skinny QR), LSQR
- Ongoing research:
  - Communication avoiding methods
  - Pipelined and s-step methods

**Ifpack2 - single-level solvers and preconditioners**

- incomplete factorisations
  - ILUT
  - RILU(k)
- relaxation preconditioners
  - Jacobi
  - Gauss-Seidel (and a multithreaded variant)
  - Successive Over-Relaxation (SOR)
  - Symmetric versions of Gauss-Seidel and SOR
  - Chebyshev
- additive Schwarz domain decomposition

Hands-on: Krylov methods and preconditioning

Go to `https://xsdk-project.github.io/MathPackagesTraining/lessons/krylov_amg/`

Sets 1 and 2

20 mins

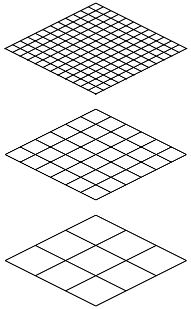
## The motivation for Multigrid methods

**Convergence of Jacobi:**
**High frequency error is damped quickly, low frequency error slowly**

## The motivation for Multigrid methods

**Convergence of Jacobi:**
**Local transmission of information cannot result in a scalable method**

## Geometric Multigrid



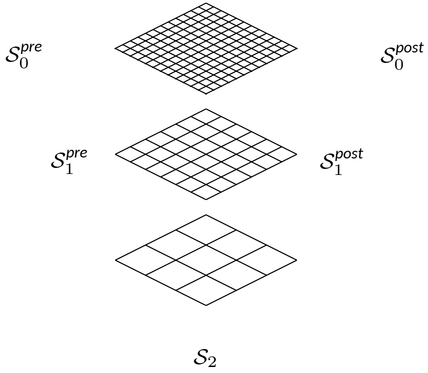
### Basic ideas

- Reconstruct the fine level solution from information of coarse representations of the fine problem.

### Observation

Low frequency on the finest level can be represented by high frequency on a coarser level.

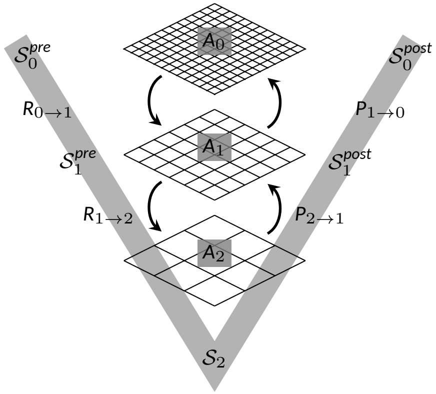
## Geometric Multigrid



### Basic ideas

- Reconstruct the fine level solution from information of coarse representations of the fine problem.
- Apply cheap **smoothers** on each multigrid level.

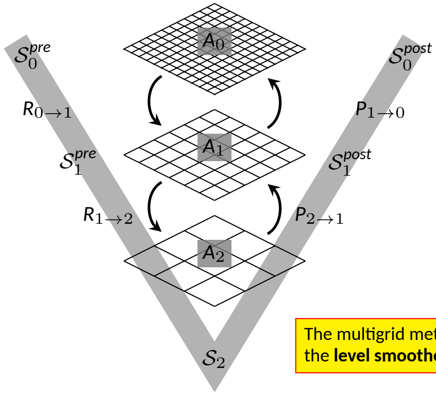
## Geometric Multigrid



### Basic ideas

- Reconstruct the fine level solution from information of coarse representations of the fine problem.
- Apply cheap **smoothers** on each multigrid level.
- **Restriction** and **prolongation operators** transfer information between different multigrid levels.

## Geometric Multigrid

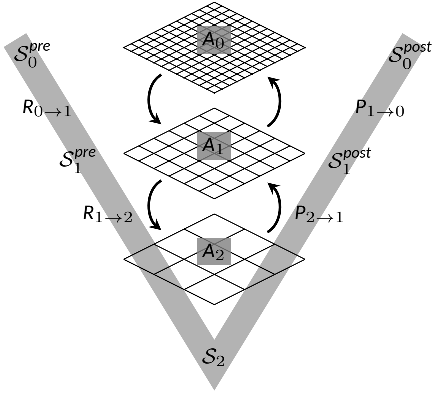


### Basic ideas

- Reconstruct the fine level solution from information of coarse representations of the fine problem.
- Apply cheap **smoothers** on each multigrid level.
- **Restriction** and **prolongation operators** transfer information between different multigrid levels.

The multigrid method is fully defined by
the **level smoothers** and **transfer operators**!

## Geometric Multigrid

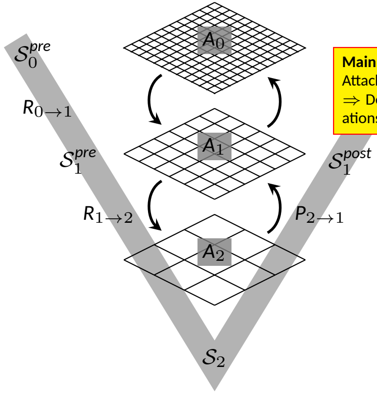


### Recursive algorithm

**Multigrid($x^k$, $b^k$):**

1. If problem is small, use direct solver.
2. Presmoothing: Apply $\mathcal{S}_k^{pre}$ on $x^k$.
3. Transfer residual $r^k$ to next coarser level:
$$r^{k+1} = R_{k\to k+1} r^k$$
4. Call Multigrid($x^{k+1}$, $r^{k+1}$).
5. Transfer correction $x^{k+1}$ to fine grid and add to $x^k$:
$$x^k = x^k + P_{k+1\to k} x^{k+1}$$
6. Postsmoothing: Apply $\mathcal{S}_k^{post}$ on $x^k$.

## Geometric Multigrid



**Main idea:**
Attack different components of the error on different grids/levels!
$\Rightarrow$ Desired optimal behaviour: convergence in a fixed number of iterations independent of problem size $n$.

### Recursive algorithm

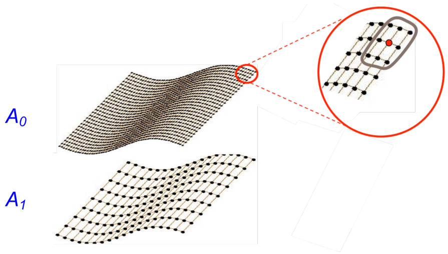
solver.

$x^k$.

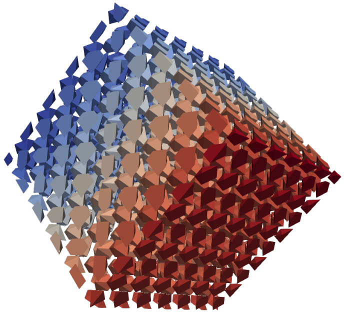
3. Transfer residual $r^k$ to next coarser level:
$$r^{k+1} = R_{k\to k+1} r^k$$
4. Call Multigrid($x^{k+1}, r^{k+1}$).
5. Transfer correction $x^{k+1}$ to fine grid and add to $x^k$:
$$x^k = x^k + P_{k+1\to k} x^{k+1}$$
6. Postsmoothing: Apply $\mathcal{S}_k^{post}$ on $x^k$.

# Algebraic Multigrid (AMG)

- Creating multigrid levels based on geometric information is not always feasible or convenient.
- Ideally, users would like to only supply their matrix ***A*** and have levels be created automatically.
- Form artificial coarse grid unknowns:
  - By selecting a subset of the fine grid unknowns (Classical AMG)
  - By grouping unknowns into “aggregates” based on connectivity in the matrix graph (Aggregation-based AMG)
- Construct transfer operators that preserve the near-nullspace of the problem.
  (And imitate the high-/low-frequency splitting of geometric multigrid.)



**Aggregates for a 2D problem**



**Aggregates for a 3D Poisson problem**

## Software packages for Algebraic Multigrid

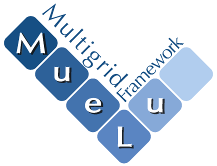
- Classical AMG (hypre)
  Developed at Lawrence Livermore National Lab.
  $\rightarrow$ **Opportunity to speak to Ulrike Meier Yang during the speed-dating.**

- Smoothed Aggregation Multigrid (PETSc)
  Developed by Mark Adams and the PETSc team.
  $\rightarrow$ **Opportunity to speak to Barry Smith during the speed-dating.**

- Smoothed Aggregation Multigrid (Trilinos)
  Two multigrid packages in Trilinos:
  - ML
    C library, up to 2B unknowns, MPI only. (Maintained, but not under active development)
  - MueLu
    Templated C++ library with support for 2B+ unknows and next-generation architectures (OpenMP, CUDA, ...)

## The MueLu package

- Robust, scalable, portable AMG preconditioning is critical for many large-scale simulations
  - Multifluid plasma simulations
  - Shock physics
  - Magneto-hydrodynamics (MHD)
  - Low Mach computational fluid dynamics (CFD)
- Capabilities
  - Aggregation-based and structured coarsening
  - Smoothers: Jacobi, Gauss-Seidel, $\ell_1$ Gauss-Seidel, multithreaded Gauss-Seidel, polynomial, ILU
  - Load balancing for good parallel performance
- Ongoing research
  - performance on next-generation architectures
  - AMG for multiphysics
  - Multigrid for coupled structured/unstructured problems
  - Algorithm selection via machine learning



Hands-on: Algebraic Multigrid

Go to `https://xsdk-project.github.io/MathPackagesTraining/lessons/krylov_amg/`

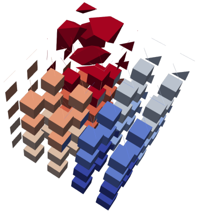
Sets 3 and 4

20 mins

## Next generation architectures and applications

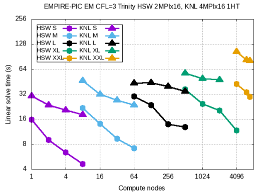
### Optimizing Multigrid Setup for Structured Grids

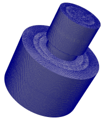
- Exploit mesh structure to speed up multigrid setup & solve.
- Stay as “algebraic” as possible.



### Multigrid for Maxwell’s equations

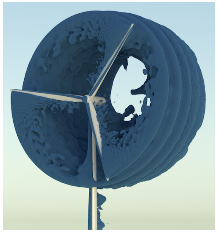
- Full Maxwell system
- Coupling with particle code
- Target architectures: Haswell, KNL, GPU
- Largest problem to date: $\sim$34B unknowns





### Multigrid for low Mach CFD

- Critical component in wind turbine simulations
- Two linear solves:
  - Momentum: GMRES/symmetric Gauss-Seidel
  - Pressure: GMRES/AMG



## Take away messages

- CG works for spd matrix and preconditioner. GMRES works for unsymmetric systems, but requires more memory.
- Simple preconditioners can reduce the number of iterations, but often do not lead to a scalable solve.
- Multigrid can lead to a constant number of iterations, independent of the problem size.

Thank you for your attention!

**Interested in working on Multigrid (and other topics) at a national lab?**

We are always looking for motivated

- summer students (LINK),
- postdocs (LINK).

Please come and talk to us !