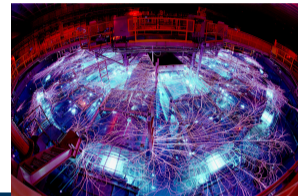


*Exceptional service in the national interest*



## Krylov Solvers and Preconditioning

Jonathan Hu and Christian Glusa, {jhu,caglusa}@sandia.gov

Presented to ATPESC 2020 Participants  
August 4, 2020



Sandia National Laboratories is a multission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

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

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

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

### Example

2D Poisson equation:

$$-\Delta u = f \text{ in } \Omega = [0, 1]^2,$$

$$u = 0 \text{ on } \partial\Omega.$$

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

$$4u_{i,j} - u_{i,j+1} - u_{i,j-1} - u_{i+1,j} - u_{i-1,j} = f(x_{i,j})\Delta x^2 \quad \text{if } x_{i,j} \notin \partial\Omega,$$

$$u_{i,j} = 0 \quad \text{if } x_{i,j} \in \partial\Omega.$$


→ 5 entries or less per row of  $\mathbf{A}$ .

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

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

$$\text{rowptr} = ( 0 \quad 2 \quad 4 \quad 5 )$$

$$\text{indices} = ( 0 \quad 1 \quad 0 \quad 1 \quad 2 )$$

$$\text{values} = ( 1 \quad 2 \quad 3 \quad 4 \quad 5 )$$


## 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.)
- Generally more restrictions on structure of  $\mathbf{A}$ .

## 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.)
- Generally more restrictions on structure of  $\mathbf{A}$ .

## Krylov methods

Based on mat-vecs, we can compute

$$\begin{aligned}\vec{y}^0 &= \vec{x}^0 && \text{("initial guess")} \\ \vec{y}^{k+1} &= \vec{y}^k + \underbrace{(\vec{b} - \mathbf{A}\vec{y}^k)}_{\text{"residual"}}\end{aligned}$$

and recombine in some smart way to obtain an approximate solution

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

Expressions for  $\alpha_k$  typically involve inner products between vectors in the so-called *Krylov space*  $\text{span} \{ \vec{y}^k \} = \{ \vec{x}^0, \mathbf{A}\vec{x}^0, \mathbf{A}^2\vec{x}^0, \mathbf{A}^3\vec{x}^0, \dots \}$ .

- 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)  $\mathbf{A}$ .
- 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

CG convergence result:

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

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

$$\kappa(\mathbf{A}) = \|\mathbf{A}\| \|\mathbf{A}^{-1}\|.$$

A common theme with Krylov methods:

$\kappa$  measures how hard it is to solve the system, i.e. how many iterations are required to reach a given tolerance.

### Idea

Reduce the condition number (“*Preconditioning*”).

Instead of solving

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

solve

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

or

$$\mathbf{A}\vec{z} = \vec{b}, \quad \vec{x} = \mathbf{P}\vec{z}$$

with *preconditioner*  $\mathbf{P}$  so that  $\kappa(\mathbf{P}\mathbf{A}) \ll \kappa(\mathbf{A})$ .

Two requirements that must be balanced:

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

## Some simple preconditioners

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



[www.trilinos.org](http://www.trilinos.org)

- Support for hybrid (MPI+X) parallelism,  $X \in \{\text{OpenMP, CUDA, ...}\}$
- C++, 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/MathPackagesTraining2020/lessons/krylov\\_amg\\_muelu/](https://xsdk-project.github.io/MathPackagesTraining2020/lessons/krylov_amg_muelu/)

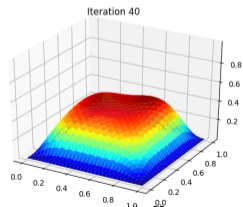
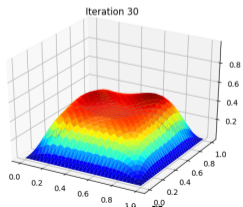
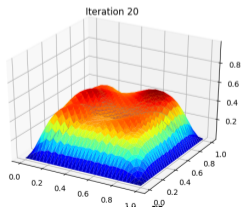
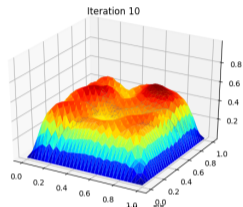
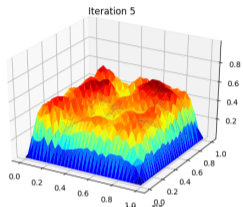
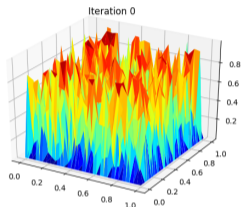
Sets 1 and 2

20 mins

# The motivation for Multigrid methods

## Convergence of Jacobi:

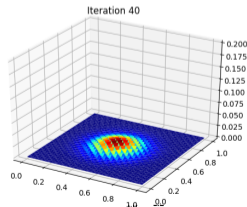
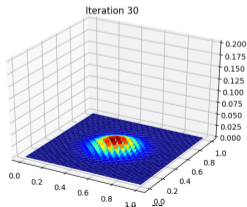
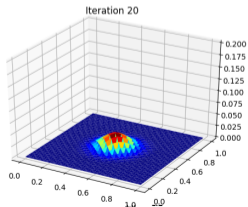
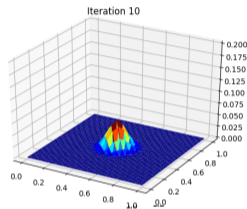
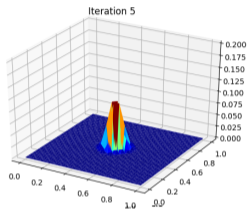
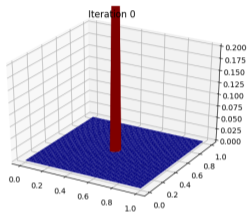
High frequency error is damped quickly, low frequency error slowly

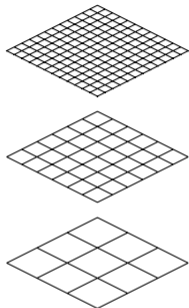


# The motivation for Multigrid methods

## Convergence of Jacobi:

Local transmission of information cannot result in a scalable method





- Main idea: accelerate solution of  $\mathbf{A}\vec{x} = \vec{b}$  by using "hierarchy" of coarser problems
- Remove high-frequency error on fine mesh, where application matrix lives (using Jacobi or another cheap preconditioner),
- Move to coarser mesh
- Remove high-frequency error on coarser mesh by solving residual equation
- Move to coarser mesh
- $\vdots$
- Solve a small problem on a very coarse mesh.
- Move back up.

Repeat.

- *Geometric multigrid* requires coarse mesh information.
- *Algebraic multigrid* constructs coarser matrices on the fly based on fine-level matrix entries.

## Software packages for Algebraic Multigrid

- Classical AMG (hypre)

Developed at Lawrence Livermore National Lab, **presentation by Ulrike Yang, 11:45 AM CDT.**



- Smoothed Aggregation Multigrid (PETSc)

Developed by Mark Adams and the PETSc team.

- 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+ unknowns and next-generation architectures (OpenMP, CUDA, ...)



# The MueLu package

- Algebraic Multigrid package in Trilinos  
Templated C++ library with support for 2B+ unknowns and next-generation architectures (OpenMP, CUDA, ...)
- 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



[www.trilinos.org](http://www.trilinos.org)

## Hands-on: Algebraic Multigrid

Go to `https://xsdk-project.github.io/MathPackagesTraining2020/lessons/krylov\_amg\_muelu/#set-3---krylov-solver-multigrid-preconditioner`

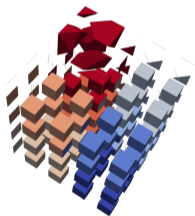
Sets 3

20 mins

# Next generation architectures and applications

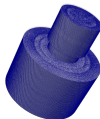
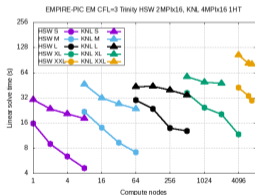
## Optimizing Multigrid Setup for Structured Grids

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



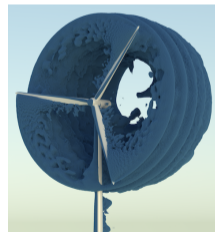
## Multigrid for Maxwell's equations

- Full Maxwell system
- Coupling with particle code
- Target architectures: Haswell, KNL, GPU
- Largest problem to date: ~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 contact us!