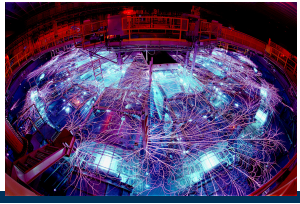
Exceptional service in the national interest

Sandia National Laboratories



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 multimission 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.

U.S. DEPARTMENT OF ENERGY | NNSA National Nuclear Security Administration

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:

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

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

$$4u_{i,j} - u_{i,j+1} - u_{i,j-1} - u_{i+1,j} - u_{i-1,j} = f(x_{i,j})\Delta x^2 \quad \text{if } x_{i,j} \notin \partial\Omega,$$
$$u_{i,j} = 0 \quad \text{if } x_{i,j} \in \partial\Omega.$$

$\rightarrow$ 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):

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

$$\texttt{rowptr} = (\,0 \;\; 2 \;\; 4 \;\; 5\,)$$
$$\texttt{indices} = (\,0 \;\; 1 \;\; 0 \;\; 1 \;\; 2\,)$$
$$\texttt{values} = (\,1 \;\; 2 \;\; 3 \;\; 4 \;\; 5\,)$$

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

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

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

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

Expressions for α_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

CG convergence result:

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

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

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

A common theme with Krylov methods:
κ 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

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

solve

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

with *preconditioner* $\boldsymbol{P}$ so that $\kappa(\boldsymbol{PA}) \ll \kappa(\boldsymbol{A})$.
Two requirements that must be balanced:

- 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.

Krylov methods and preconditioners: Packages in the Trilinos project

www.trilinos.org

- Support for hybrid (MPI+X) parallelism, $X \in \{\text{OpenMP, CUDA, } \ldots\}$
- 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

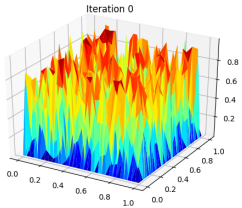
Go to `https://xsdk-project.github.io/MathPackagesTraining2020/lessons/krylov_amg_muelu/`

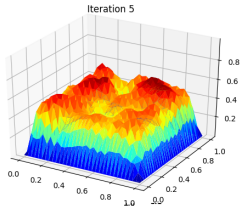
Sets 1 and 2

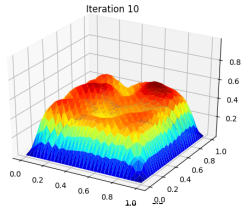
20 mins

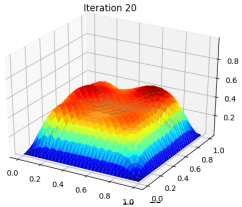
The motivation for Multigrid methods

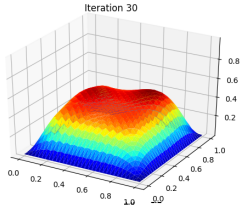
Convergence of Jacobi:
High frequency error is damped quickly, low frequency error slowly

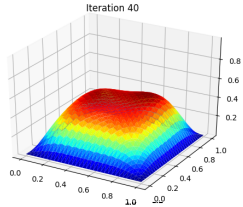






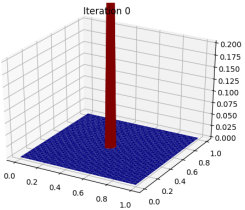


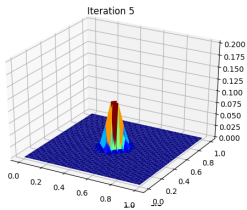


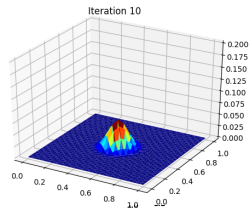


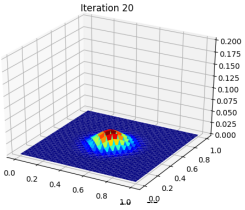
The motivation for Multigrid methods

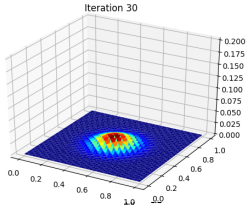
Convergence of Jacobi:
Local transmission of information cannot result in a scalable method

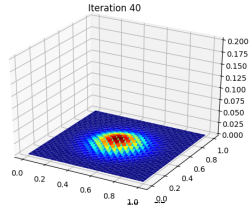




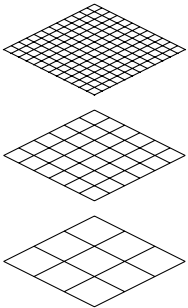








Multigrid



- Main idea: accelerate solution of $\boldsymbol{A}\vec{x} = \vec{b}$ 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

  ⋮
- 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+ unknows 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, ℓ_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

Hands-on: Algebraic Multigrid

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

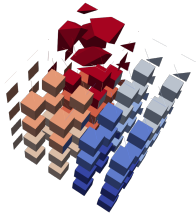
Sets 3

20 mins

Next generation architectures and applications

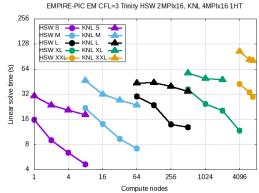
Optimizing Multigrid Setup for Structured Grids

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



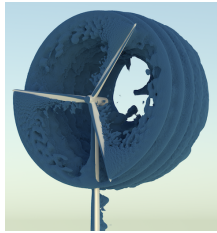
Multigrid for Maxwell’s equations

- 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!