Krylov Solvers and Preconditioning

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

Presented to ATPESC 2020 Participants
August 4, 2020
Discretization of partial differential equations gives rise to large linear systems of equations

\[ A\vec{x} = \vec{b}, \]

where \( A \) 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 \( \{x_{i,j}\} \):**

\[ 4u_{i,j} - u_{i,j+1} - u_{i,j-1} - u_{i+1,j} - u_{i-1,j} = f(x_{i,j}) \Delta x^2 \]
\[ u_{i,j} = 0 \]

if \( x_{i,j} \not\in \partial\Omega, \)

if \( x_{i,j} \in \partial\Omega \).

→ 5 entries or less per row of \( A \).

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

\[
A = \begin{pmatrix}
1 & 2 & 0 \\
3 & 4 & 0 \\
0 & 0 & 5
\end{pmatrix}
\]

\[
\text{rowptr} = (0 \ 2 \ 4 \ 5) \]

\[
\text{indices} = (0 \ 1 \ 0 \ 1 \ 2)
\]

\[
\text{values} = (1 \ 2 \ 3 \ 4 \ 5)
\]
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

\[ A\vec{x} = \vec{b}. \]

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

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

**Observation**

\( A \) 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 \( A \).
Krylov methods

Based on mat-vecs, we can compute

\[
\vec{y}^{0} = \vec{x}^{0}
\]

(“initial guess”)

\[
\vec{y}^{k+1} = \vec{y}^{k} + \left( \vec{b} - A\vec{y}^{k} \right)
\]

“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 \(\alpha_{k}\) typically involve inner products between vectors in the so-called Krylov space

\[
\text{span} \left\{ \vec{y}^{k} \right\} = \left\{ \vec{x}^{0}, A\vec{x}^{0}, A^{2}\vec{x}^{0}, A^{3}\vec{x}^{0}, \ldots \right\}.
\]

- 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) \(A\).

- 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:

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

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

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

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

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

solve

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

with preconditioner $P$ so that $\kappa(PA) \ll \kappa(A)$.

Two requirements that must be balanced:

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

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

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

Next generation architectures and applications
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!