Iterative Solvers & Algebraic Multigrid (with Trilinos, Belos & MueLu)

Christian Glusa and Graham Harper  \{caglusa,gbharpe\}@sandia.gov

Presented to ATPESC 2022 Participants
August 9, 2022

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. SAND NO SAND2022-10602 C

1 / 17
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**

<table>
<thead>
<tr>
<th>2D Poisson equation:</th>
<th>Central finite differences on a uniform mesh ( {x_{i,j}} ):</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-\Delta u = f ) in ( \Omega = [0,1]^2 ), ( u = 0 ) on ( \partial \Omega ).</td>
<td>( 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 ) if ( x_{i,j} \notin \partial \Omega ), ( u_{i,j} = 0 ) if ( x_{i,j} \in \partial \Omega ).</td>
</tr>
</tbody>
</table>

→ 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

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

Option 1: Direct solvers (think Gaussian elimination), presentation by Sherry Li, and Pieter Ghysels in the other room

- Factorisation scales as \( O(n^3) \).
- 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 \( O(n) \) non-zero entries. \( \rightarrow \) Optimal complexity for a solve is \( O(n) \) operations.

Option 2: Iterative solvers

- Exploit an operation that has \( O(n) \) 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 properties of \( \mathbf{A} \).
Available solvers

Solve

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

Option 1: Direct solvers (think Gaussian elimination), presentation by Sherry Li, and Pieter Ghysels in the other room

- Factorisation scales as \( O(n^3) \).
- 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 \( O(n) \) non-zero entries. \( \rightarrow \) Optimal complexity for a solve is \( O(n) \) operations.

Option 2: Iterative solvers

- Exploit an operation that has \( O(n) \) 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 properties of \( A \).
Krylov methods

Based on mat-vecs, we can compute

\[ \vec{y}^0 = \vec{x}^0 \]  
\[ \vec{y}^{k+1} = \vec{y}^k + (\vec{b} - A\vec{y}^k) \]

(“initial guess”’)

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 span \( \{\vec{y}^k\} = \{\vec{x}^0, A\vec{x}^0, A^2\vec{x}^0, A^3\vec{x}^0, \ldots\} \).

- 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 nonsymmetric systems.
  - GMRES keeps the whole Krylov space around.
  - GMRES(\( K \)) discards the Krylov space after \( K \) iterations.
Convergence of Krylov methods

CG convergence result:

\[
\left\| \bar{x}^K - \bar{x} \right\| \leq \left( 1 - 1/\sqrt{\kappa(A)} \right)^K \left\| \bar{x}_0 - \bar{x} \right\|,
\]

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

\[
\kappa(A) = \left\| A \right\| \left\| A^{-1} \right\|.
\]

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

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

solve

\[
PA\bar{x} = \bar{P}\bar{b}
\]

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: \( \mathbf{P} = \mathbf{D}^{-1} \), 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.
Krylov methods and preconditioners: Packages in the Trilinos project

- Support for hybrid (MPI+X) parallelism, \( X \in \{\text{OpenMP, CUDA, HIP, \ldots}\} \)
- C++, open source, primarily developed at Sandia National Labs

**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
  - Mixed precision 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/MathPackagesTraining2022/lessons/krylov_amg_muelu/
Sets 1 and 2
20 mins
Slack channel: #track5-numerical
The motivation for Multigrid methods

Convergence of Jacobi: \( \vec{y}^{k+1} = \vec{y}^k + D^{-1}\vec{r}^k, \quad \vec{r}^k = \vec{b} - A\vec{y}^k \)

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 \( A\vec{x} = \vec{b} \) 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 Sarah Osborn & Ulrike Yang, 3:15 PM.

[Image: hypre]

- 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, HIP, ...)

[Image: MueLu]
The MueLu package

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

- 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/MathPackagesTraining2022/lessons/krylov_amg_muelu/
Set 3 & 4
20 mins
Slack channel: #track5-numerical
Strong & weak scaling results for EMPIRE (Maxwell + PIC)

- Specialized multigrid for curl-curl problem
- Largest problem to date: 34B unknowns
Ongoing work

- Multiprecision (Krylov methods with mixed precision; lower precision preconditioning)
- Multigrid approaches for higher order discretizations
- Matrix-free multigrid
- Multigrid on semi-structured meshes
- Machine learning for AMG coarsening
- Preconditioning for multiphysics systems
- Multigrid for hierarchical matrices (boundary integral and nonlocal equations)
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 solver.
- Multigrid (when applicable) has 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!