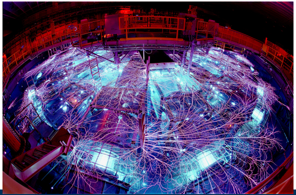
Sandia National Laboratories



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

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

August 8, 2023

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

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. SAND2023-07418C

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 $\{x_{i,j}\}$:

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

$\rightarrow$ 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):

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

$$\texttt{rowptr} = \begin{pmatrix} 0 & 2 & 4 & 5 \end{pmatrix}$$

$$\texttt{indices} = \begin{pmatrix} 0 & 1 & 0 & 1 & 2 \end{pmatrix}$$

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

Available solvers

Solve

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

Option 1: Direct solvers (think Gaussian elimination), **presentation by Sherry Li, and Pieter Ghysels this morning**

- 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 properties of $\boldsymbol{A}$.

Available solvers

Solve

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

Option 1: Direct solvers (think Gaussian elimination), **presentation by Sherry Li, and Pieter Ghysels this morning**

- 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 properties of $\boldsymbol{A}$.

Krylov methods

Based on mat-vecs, we can compute

$$\vec{y}^0 := \vec{b} \qquad \text{("initial guess"')}$$

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

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* $\operatorname{span}\left\{\vec{y}^k\right\} = \left\{\vec{b}, \boldsymbol{A}\vec{b}, \boldsymbol{A}^2\vec{b}, \boldsymbol{A}^3\vec{b}, \ldots, \boldsymbol{A}^K\vec{b}\right\}$.

- 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 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:

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

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

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

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

TRILINOS

`www.trilinos.org`

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

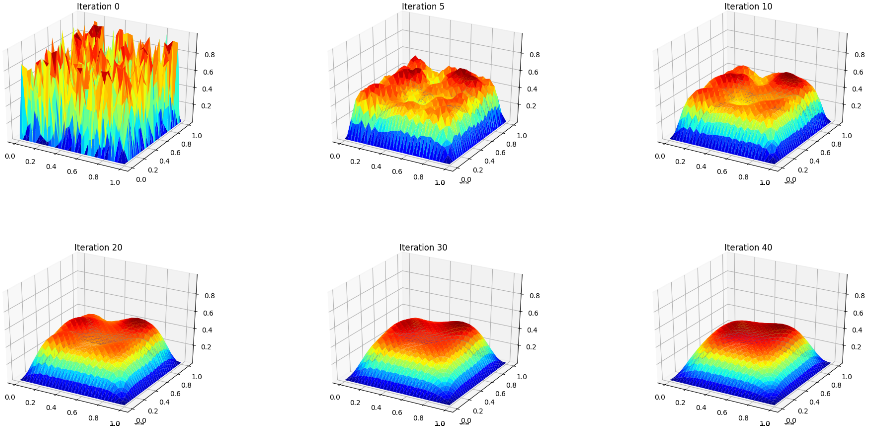
- incomplete factorisations
  - ILUT
  - RILU(k)
- relaxation preconditioners
  - Jacobi
  - Gauss-Seidel (and a multithreaded variant)
  - Successive Over-Relaxation (SOR)
  - Symmetric versions of Gauss-Seidel and SOR
  - Chebyshev
- additive Schwarz domain decomposition

Hands-on: Krylov methods and preconditioning
Go to `https://xsdk-project.github.io/MathPackagesTraining2023/lessons/krylov_amg_muelu/`
Sets 1 and 2
20 mins
Slack channel: #atpesc-2023-track5-numerical-breakout

Motivation for Multigrid methods

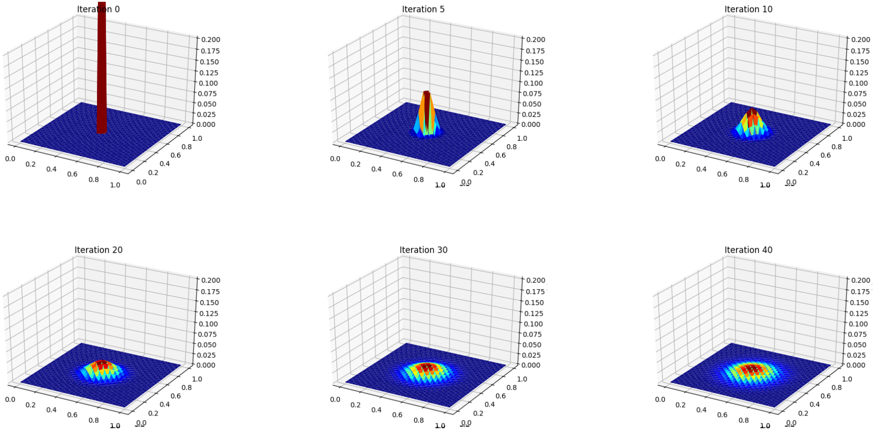
Sandia National Laboratories

Convergence of Jacobi: $\vec{y}^{k+1} = \vec{y}^{k} + \boldsymbol{D}^{-1}\vec{r}^{k}, \quad \vec{r}^{k} = \vec{b} - \boldsymbol{A}\vec{y}^{k}$
High frequency error is damped quickly, low frequency error slowly



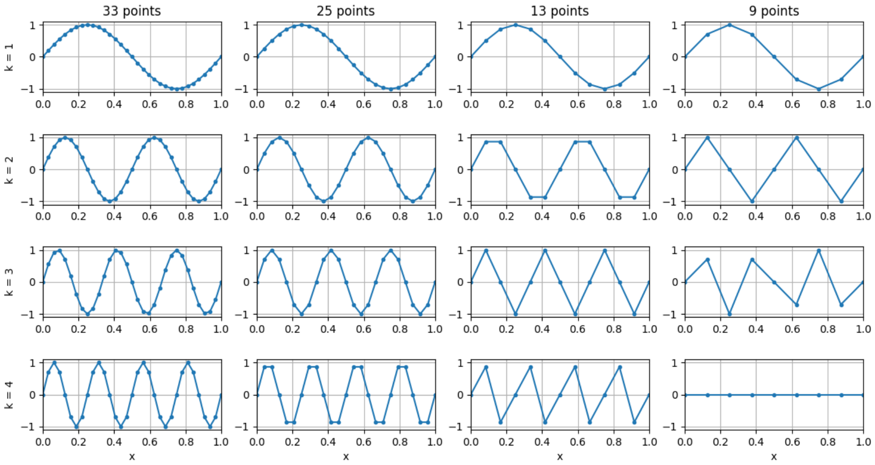
Motivation for Multigrid methods

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



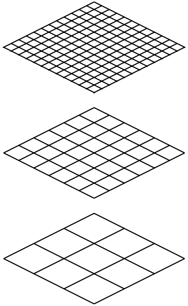
Motivation for Multigrid methods

Resolution affects observed frequency:



Idea: accelerate Jacobi convergence by reducing resolution!

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

  $\vdots$

- 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 Sarah Osborn & Ulrike Yang this morning**.
- 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, ...)

The MueLu package

Sandia National Laboratories

- 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, 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/MathPackagesTraining2023/lessons/krylov_amg_muelu/`

Set 3 & 4

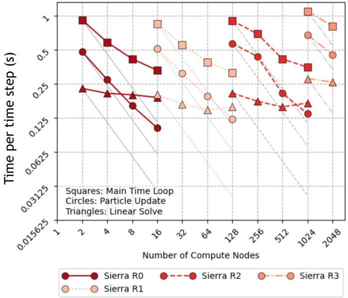
20 mins

Slack channel: #atpesc-2023-track5-numerical-breakout

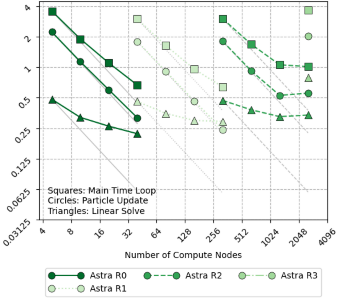
Strong & weak scaling results for EMPIRE (Maxwell + PIC)

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

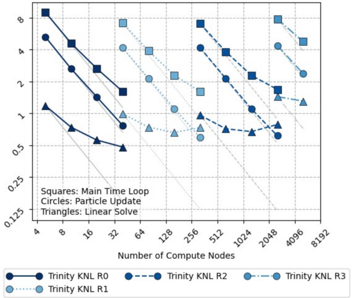
Sierra
4 GPU/socket; 1 MPI/GPU
Up to 2048 nodes (47% of total)



Astra
2 MPI/socket; 9 threads/MPI
Up to 2560 nodes (99% of total)



Trinity/KNL
4 MPI/socket; 16 threads/MPI
Up to 5120 nodes (52% of total)



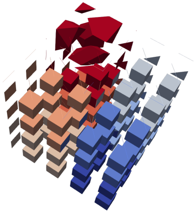
Mesh	Elements	Nodes	Edges	Particles
R0	3.7M	660k	4.4M	360M
R1	25M	4.4M	30M	2.4B
R2	200M	32M	240M	19B
R3	1.6B	270M	1.9B	160B

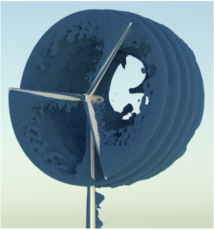
Ongoing work

- 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)

Algorithm 1 Iterative Refinement with GMRES Error Correction

$r_0 = b - Ax_0$ [double]
for $i = 1, 2, \ldots$ until convergence: **do**
  Use GMRES(m) to solve $Au_i = r_i$ for correction u_i [single]
  $x_{i+1} = x_i + u_i$ [double]
  $r_{i+1} = b - Ax_{i+1}$ [double]
end for





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).
- Sustainable Research Pathways (LINK)

Please contact us!