中国科学院大学
夏季强化课程
2022

Fast Solvers for Large Algebraic Systems

Lecture 2. Fast solvers for sparse linear systems

2 稀疏线性解法器

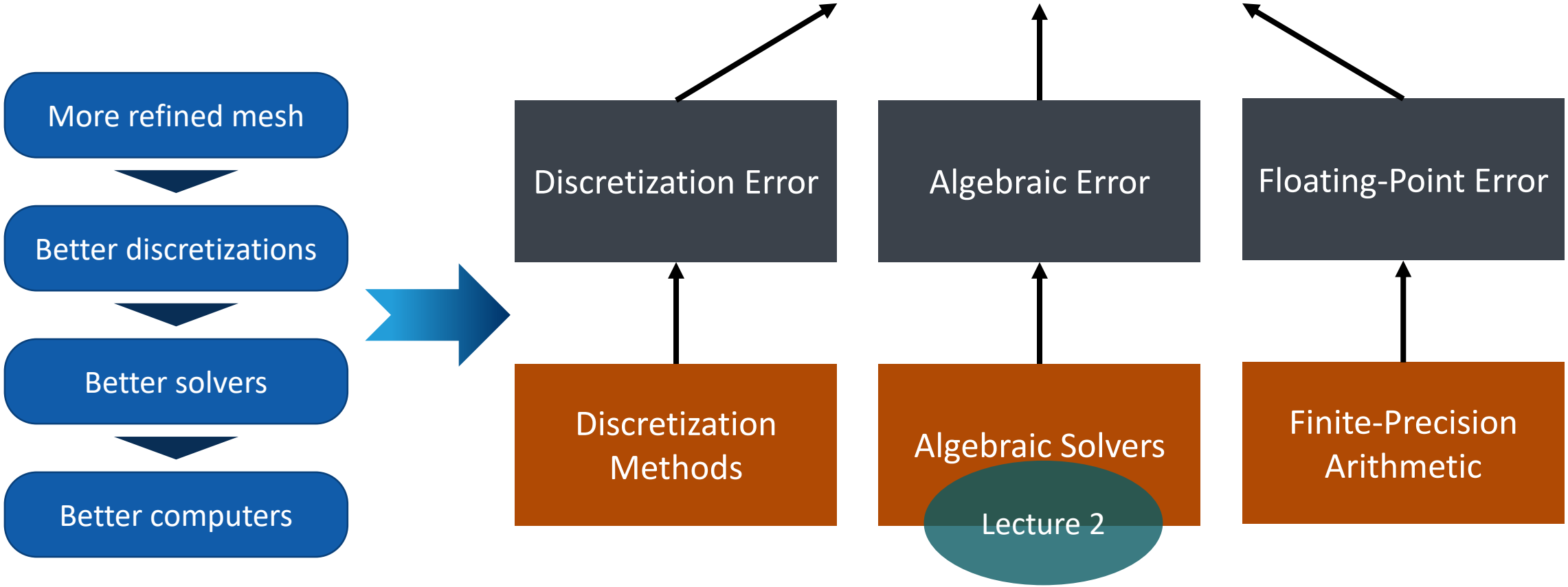
Chensong Zhang, AMSS

http://lsec.cc.ac.cn/~zhangcs

Table of Contents

Sources of Error in Simulation

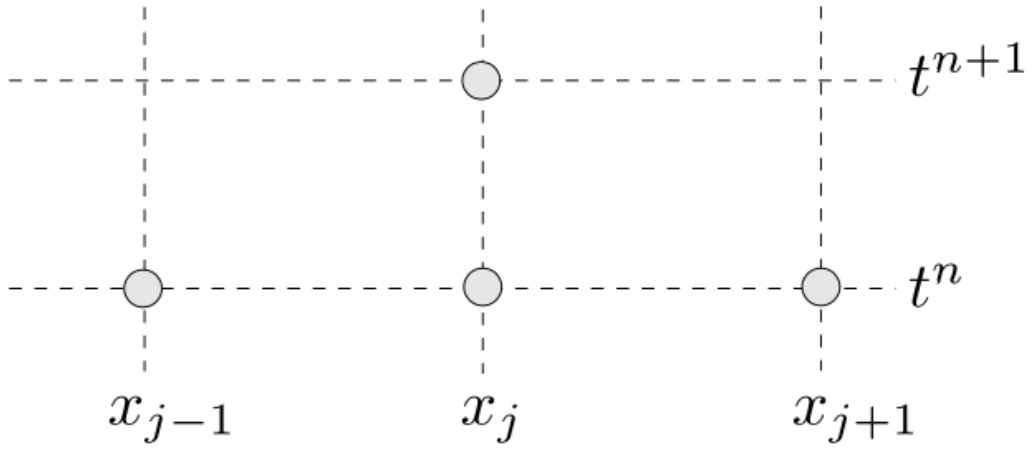
Approximation: $u(x) = U_h(x) + \mathcal{E}_{\text{dis}} + \mathcal{E}_{\text{alg}} + \mathcal{E}_{\text{fp}}$



Introduction

Solution methods for sparse linear systems

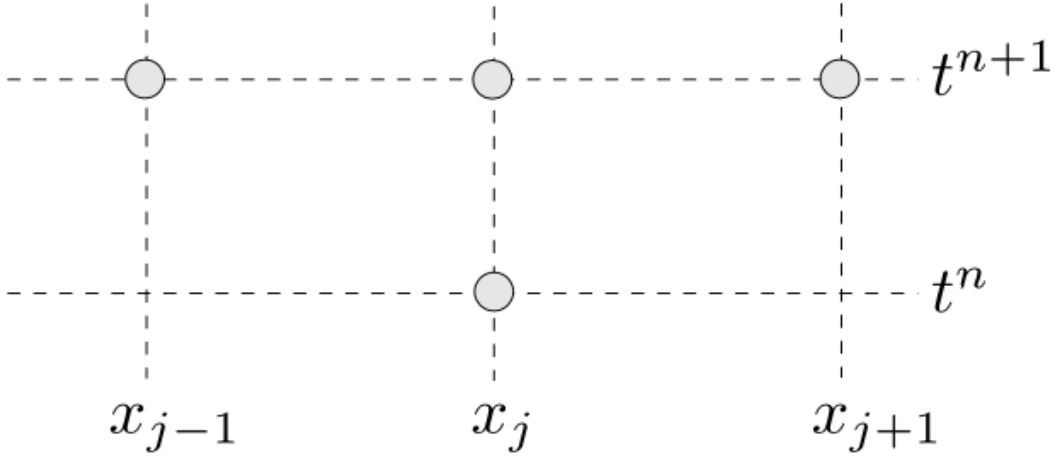
Explicit vs Implicit Methods



Explicit Discretizations

- Conditionally stable, small time stepsize
- Easy to implement
- Easy to parallelize
- No need to solve large linear systems

Strong Scalability?



Implicit Discretizations

- Unconditionally stable, larger time stepsize
- More difficult to implement
- More difficult to parallelize
- Need sparse numerical linear algebra

Weak Scalability?

Question: Should we use explicit or implicit methods? It depends.

Linear Solution Methods

Given a large sparse matrix $A \in \mathbb{R}^{N\times N}$ and $\vec{f} \in \mathbb{R}^N$, find $\vec{u} \in \mathbb{R}^N$ such that $A\vec{u} = \vec{f}$

- It looks easy! Does it?

  It is relatively easy if you don't care about performance!

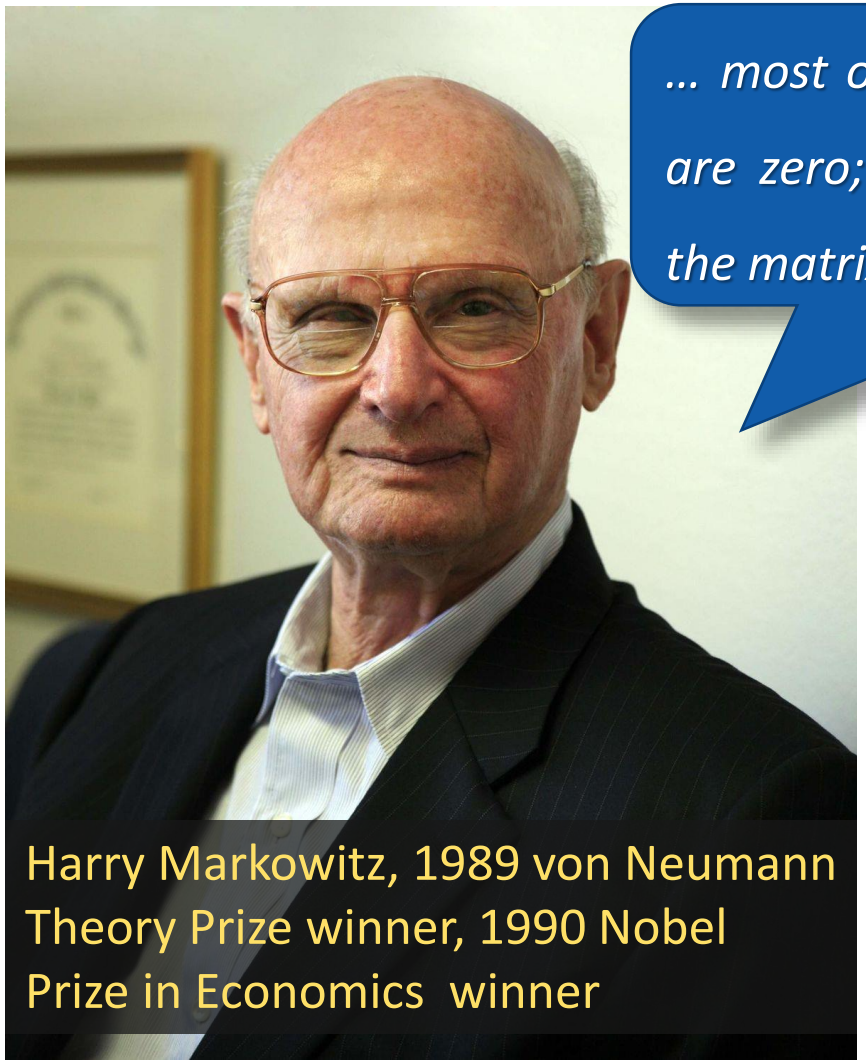
  - Large number of unknowns
  - Sometimes ill-conditioned
  - PDE-system with multiple physical variables ➔ different algebraic properties
  - Linear solution methods are difficult to scale (optimality and parallel scalability)
  - Linear solution methods are usually the bottleneck in implicit simulation

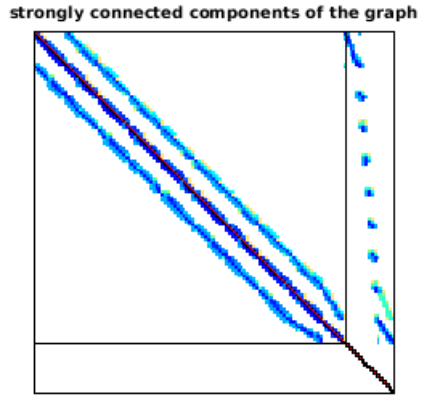
  Review complexity and HPC paradox in Lecture 1

- Design goals of linear solvers:

  accuracy, convergence, applicability, efficiency, optimality, user-friendliness, robustness, scalability, reliability, resilience, cost-effectiveness, ...

Sparse Linear Systems





https://sparse.tamu.edu/Goodwin/Goodwin_010

- Sparse matrix: number of nonzeros (nnz's) $\sim O(N)$
- Typical in numerical PDE, graph, page rank, economy, …
- How to store sparse matrices?

  CSR, CSRx, CSC, DIA, ELL, COO, HYB, CSR5, …
- How to perform operations on sparse matrices?
- SpMV, SpGEMM, SpMM (Sparse*Dense)?

Sparse Matrix Formats

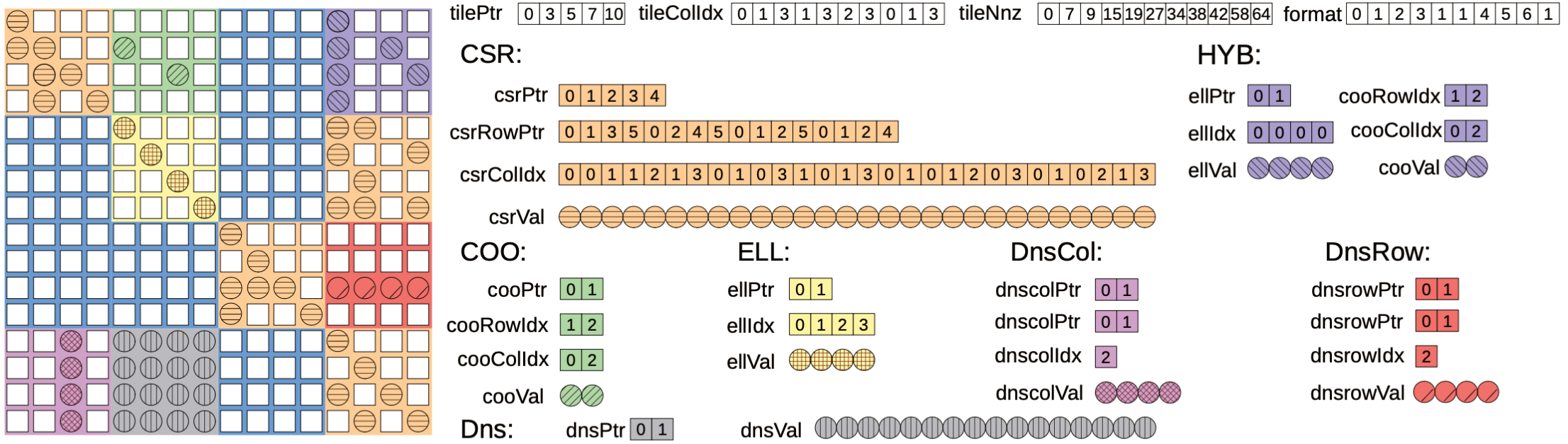
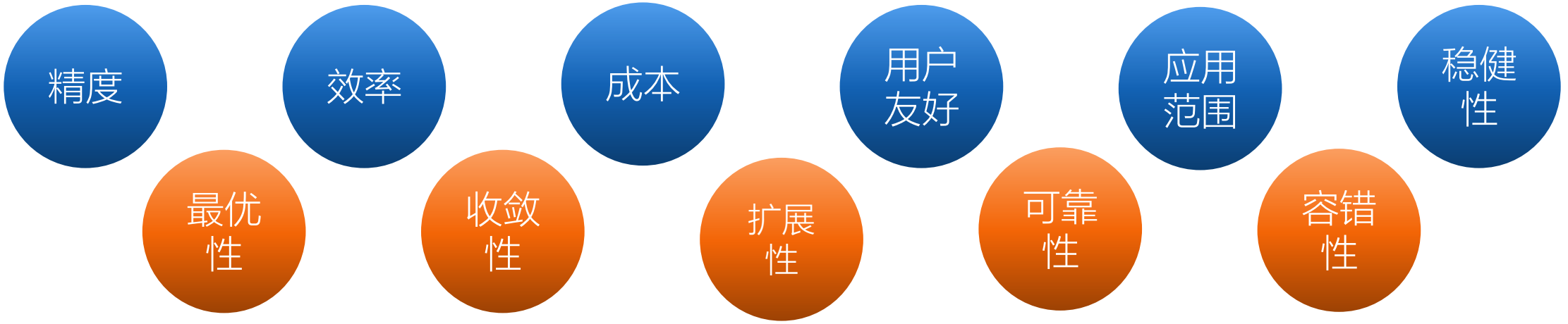


Fig. 3. An example matrix A of size 16-by-16 stored in 10 sparse tiles of size 4-by-4. The tile structure includes three arrays tilePtr, tileColIdx and tileNnz representing the memory offsets of tiles, tile column indices and the offsets for the number of nonzeros in sparse tiles. According to the format selection method, four of the 10 tiles keep the CSR format unchanged and use three arrays to store their information. The remaining six tiles are transformed to different formats, including COO, ELL, HYB, Dns, DnsRow and DnsCol. Each format has several corresponding arrays to store the nonzeros and their indices.

Source: Y. Niu, Z. Lu, M. Dong, Z. Jin, W. Liu and G. Tan, "TileSpMV: A Tiled Algorithm for Sparse Matrix-Vector Multiplication on GPUs," 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2021, pp. 68-78, doi: 10.1109/IPDPS49936.2021.00016

Method of Choice



The Method of Choice Award

- Iain Duff: "*Real men use direct methods*."
- Use the direct method whenever possible (moderate-size problems)
- Standard iterative methods work just fine, but don't count on them
- Achi Brandt: "*The optimal method might not be the fastest!*"
- Matrix-free implementation is preferred whenever applicable
- Take-home: **Think about the solution method as early as possible**

Direct Solvers vs Iterative Solvers

Direct solvers < Iterative solvers

- Optimality: optimal complexity is possible, $O(N)$ operations
- Effectiveness: adjustable accuracy with good initial guess in practice
- Efficiency: matrix-free operations can be used in some applications
- Applicability: singular or nearly-singular problems can be solved efficiently

Direct solvers > Iterative solvers

- Speed: optimal algorithm might not be the fastest algorithm
- Problem-dependence: require different methods for different problems
- Implementation: difficult if not impossible to make a general-purpose package
- Multiple RHS: same coefficient matrix (or local updates) and many right-hand sides
- Robustness: biggest disadvantage in practice

Solver-Friendly Methods

模型选择

使用简单模型（更容易计算或计算量更小的模型），可计算建模

O1

离散方法

一些离散方法产生的代数方程有较好的性质，更易求解，或者已有高效求解方法

O2

网格剖分

结构化网格或者一致加密网格，可以帮助解法器提高算法效率和实现效率

O3

简化模型

显式方法

一致网格

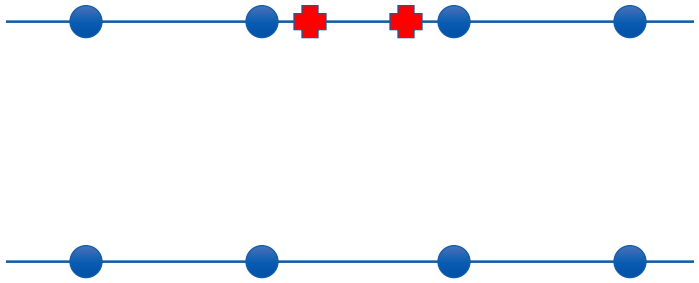
为什么不使用这样的方法呢？

有时候，身不由己……

Eulerian-Lagrangian Method

Advection-Diffusion Equation:

$$u_t + b \cdot \nabla u - \mu \Delta u = 0$$



MMOC: Douglas and Russell 1982; Pironneau 1982.

ELLAM: Celia et al. 1990; Russell 1990.

Adaptivity: Chen and Ji 2006; Jia, Hu, Xu, Z. 2010.

$$\frac{d}{dt} y(x, s; t) = b(y(x, s; t), t), \quad y(x, s; s) = x.$$

Material derivative ➔ Characteristics

$$\left(\frac{u^{k+1} - u_*^k}{\Delta t}, v \right) + \mu \left(\nabla u^{k+1}, \nabla v \right) = (f, v), \quad \forall v \in V$$

Gain: Only need to solve a standard heat equation discrete problem (SPD); solvers available!

Cost: Implementation cost, accuracy, numerical diffusion, stability, 3D unstructured grid, …

Projection Method

Navier-Stokes Equation:

$$u_t + (u \cdot \nabla)u - \mu \Delta u + \nabla p = 0$$

$$\nabla \cdot u = 0, \quad u(x,0) = u_0(x), \quad u|_{\partial\Omega} = g$$

Step 1. Compute intermediate velocity

$$\frac{1}{\Delta t}(u^* - u^k) - \mu \Delta u^* = -(u^k \cdot \nabla)u^k$$

$$u^*|_{\partial\Omega} = g$$

Step 2. Projection to divergence-free space

$$u^* = u^{k+1} + \Delta t \, \nabla p^{k+1}$$

$$\nabla \cdot u^{k+1} = 0, \quad u^{k+1} \cdot \mathbf{n}|_{\partial\Omega} = g \cdot \mathbf{n}$$

Ref: Chorin 1967; Tamam 1968

Step 2 is the Hodge decomposition

- solenoidal u^{k+1}（无源场）
- irrotational ∇p^{k+1}（无旋场）

● Step 2.1: BVP for pressure

$$-\nabla \cdot \nabla p^{k+1} = -\frac{1}{\Delta t} \nabla \cdot u^*$$

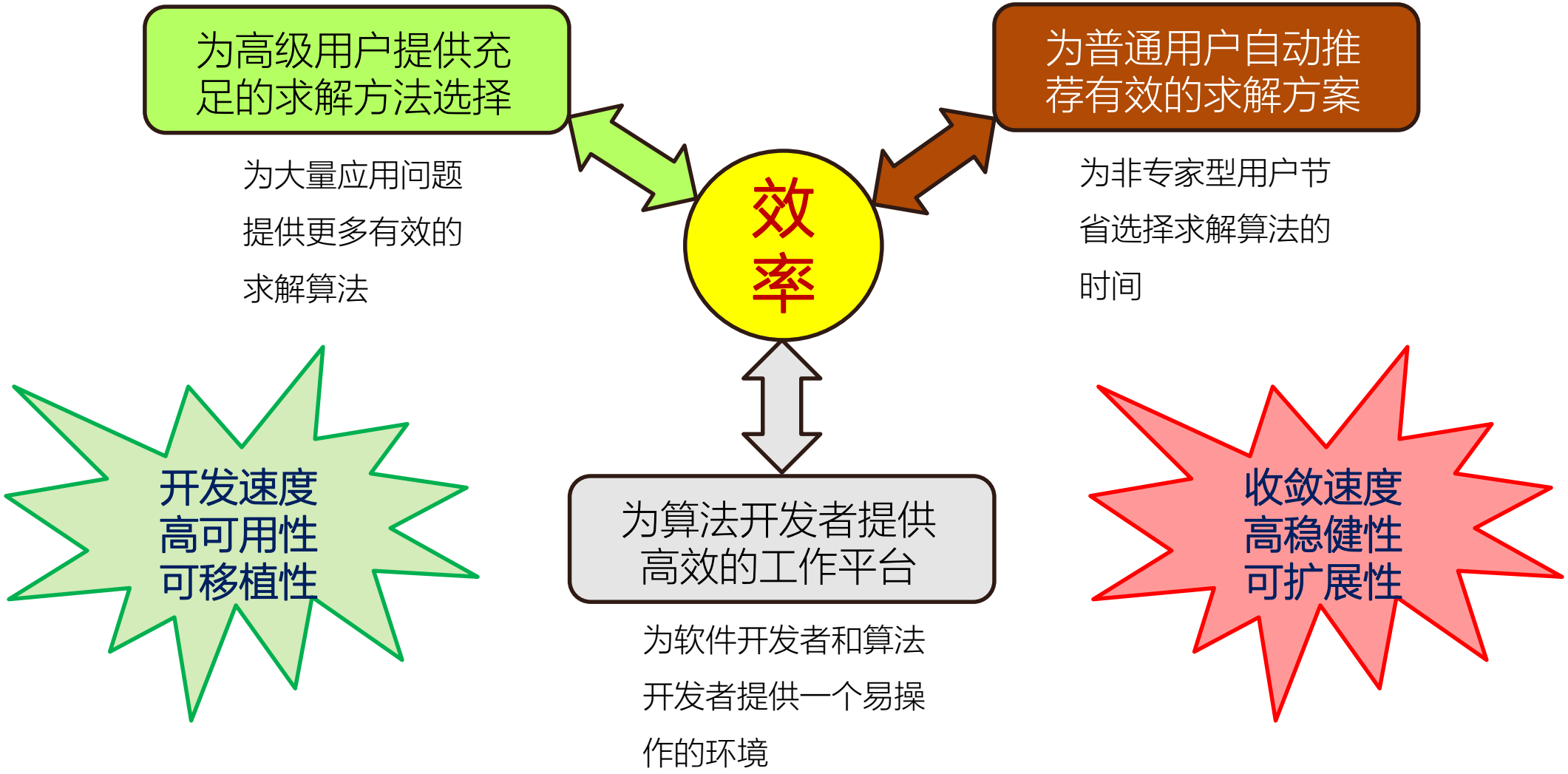
$$\left.\frac{\partial p}{\partial n}\right|_{\partial\Omega} = 0$$

● Step 2.2 Velocity correction

$$u^{k+1} = u^* - \Delta t \nabla p^{k+1}$$

$$u^{k+1} \cdot \mathbf{n}|_{\partial\Omega} = g \cdot \mathbf{n}$$

User-Friendly Solvers



Thomas Method, A Special Solver

- Thomas method for tri-diagonal systems（追赶法）

GE for tridiagonal matrix

$$\begin{pmatrix} b_1 & c_1 & 0 & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & 0 & 0 \\ 0 & 0 & a_4 & b_4 & c_4 & 0 \\ 0 & 0 & 0 & a_5 & b_5 & c_5 \\ 0 & 0 & 0 & 0 & a_6 & b_6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & \gamma_1 & 0 & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & 0 & 0 \\ 0 & 0 & a_4 & b_4 & c_4 & 0 \\ 0 & 0 & 0 & a_5 & b_5 & c_5 \\ 0 & 0 & 0 & 0 & a_6 & b_6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} \rho_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \end{pmatrix}$$

$$\begin{pmatrix} 1 & \gamma_1 & 0 & 0 & 0 & 0 \\ 0 & 1 & \gamma_2 & 0 & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & 0 & 0 \\ 0 & 0 & a_4 & b_4 & c_4 & 0 \\ 0 & 0 & 0 & a_5 & b_5 & c_5 \\ 0 & 0 & 0 & 0 & a_6 & b_6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} \rho_1 \\ \rho_2 \\ r_3 \\ r_4 \\ r_5 \\ r_6 \end{pmatrix} \cdots \begin{pmatrix} 1 & \gamma_1 & 0 & 0 & 0 & 0 \\ 0 & 1 & \gamma_2 & 0 & 0 & 0 \\ 0 & 0 & 1 & \gamma_3 & 0 & 0 \\ 0 & 0 & 0 & 1 & \gamma_4 & 0 \\ 0 & 0 & 0 & 0 & 1 & \gamma_5 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} \rho_1 \\ \rho_2 \\ \rho_3 \\ \rho_4 \\ \rho_5 \\ \rho_6 \end{pmatrix}$$

Source: W. T. Lee http://www.industrial-maths.com/ms6021_thomas.pdf

FFT-based Method, Another Special Solver

- FFT-based fast Poisson solver

$$-\Delta u(x,y) = f(x,y) \xrightarrow{FFT} -(k_x^2+k_y^2)\hat{u}(k_x,k_y) = \hat{f}(k_x,k_y)$$

Apply 2D inverse FFT to $\hat{u}(k_x,k_y)$ to obtain $u(x,y)$

Table: Kernel time (seconds) in 2D case

DOF	FFTW	FMG(1,2)	CUFFT	FMG(1,2)
1M	0.260	0.108	0.0110	0.0088
4M	2.020	0.452	0.0408	0.0257
16M	6.650	1.830	0.1364	0.0917

- Nearly optimal: $O(\log n)$ per grid point with n be the number of grid in each direction
- Limited applicability: equation, boundary condition, grid

Source: Feng, C., Shu, S., Xu, J., & Zhang, C. (2014). Numerical Study of Geometric Multigrid Methods on CPU-GPU Heterogeneous Computers. *Advances in Applied Mathematics and Mechanics, 6*(1), 1-23. doi:10.1017/S2070073300002411

Sparse Direct Solvers

Direct methods for sparse linear systems

/02

Direct Solvers

- General LU factorizations: Available in LINPACK ➔ LAPACK ➔ ScaLAPACK ➔ PLASMA

$$Ax = b \quad \Rightarrow \quad PAQ = LU \quad \Rightarrow \quad Ly = Pb, \;\; UQ^T x = y$$

Setup Solve

- Accurate (in some sense), robust, multiple RHS, ...
- Memory cost: Require explicit matrices, need a lot of RAM for decomposition
- High computational cost:

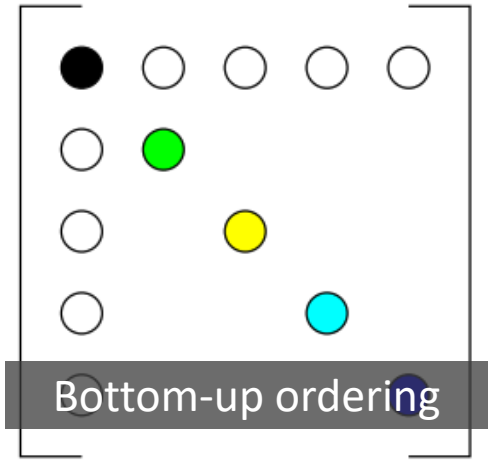
  General $O(N^3)$, banded $O(N^2)$, nested dissection $O(N^{1.5})$ for 2D 5-point stencil

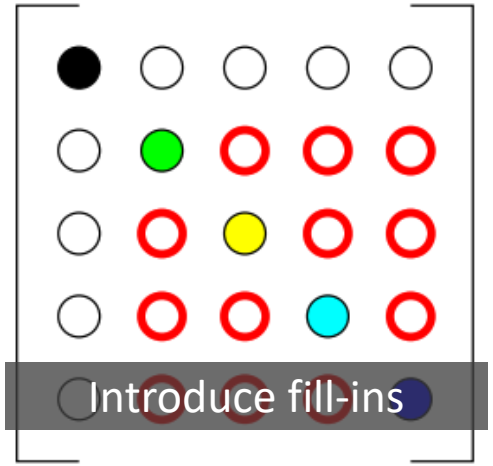
- Sparse direct solvers: The key ingredient is to preserve sparsity and maintain stability
- The permutations determine the ordering of eliminations
- Modern variants: frontal methods, super-nodal methods, multifrontal methods, ...

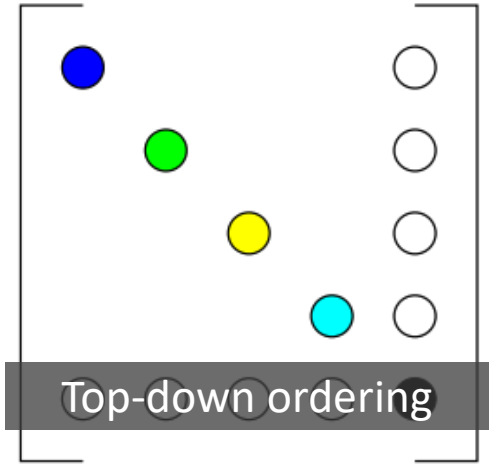
Ref: Duff, Erisman, Reid 1986; Demmel 1997; Davis 2006

Ordering and Complexity

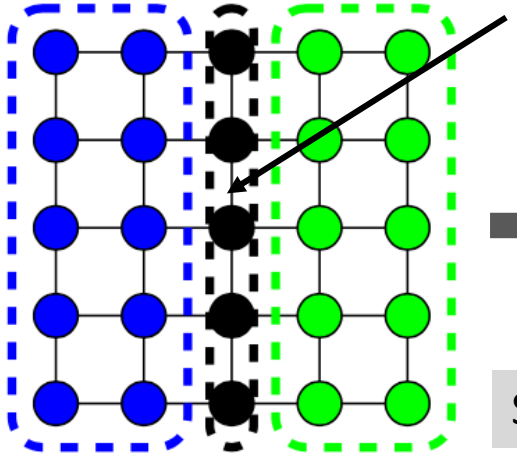
- Simple Example of Ordering


Bottom-up ordering


Introduce fill-ins


Top-down ordering

- Nested Dissection



Find a separator, and do it recursively

Grid dimensions	Matrix order	Work to factorize	Factor storage
$k \times k$	k^2	k^3	$k^2 \log k$
$k \times k \times k$	k^3	k^6	k^4

Divide-and-Conquer

Source: https://www.cs.cornell.edu/~bindel/class/cs5220-s10/

Ref: George 1973; Amestoy, Davis, Duff 2004; Bindel, Sparse Direct Solvers, 2010

Ordering and Stability

- Standard LU factorization

$$A = \begin{bmatrix} \epsilon & -1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \epsilon^{-1} & 1 \end{bmatrix} \begin{bmatrix} \epsilon & -1 \\ 0 & 1+\epsilon^{-1} \end{bmatrix}$$

- Consider fix-precision arithmetic and if $0 < \epsilon \ll 1$ is very small

$$U_{22} = 1 - L_{21}U_{12} = 1 + \epsilon^{-1} \approx \epsilon^{-1} = \hat{U}_{22}$$

- Error could be large even for this well-conditioned matrix

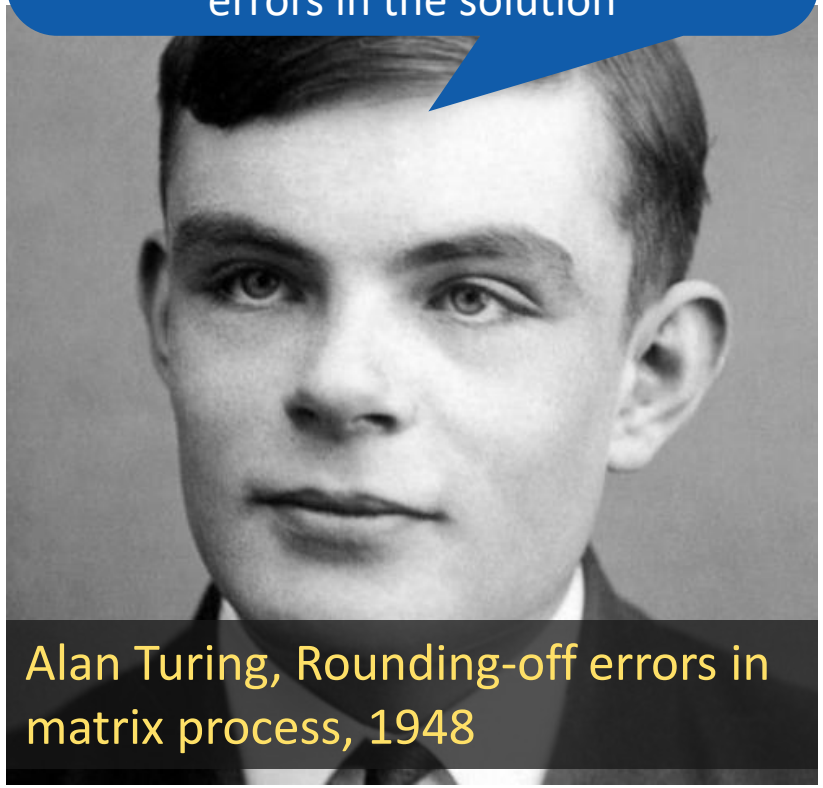
$$A - \hat{L}\hat{U} = \begin{bmatrix} \epsilon & -1 \\ 1 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ \epsilon^{-1} & 1 \end{bmatrix} \begin{bmatrix} \epsilon & -1 \\ 0 & \epsilon^{-1} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

- Use partial or complete pivoting to improve stability

Ref: Higham, N. J. "Accuracy and stability of numerical algorithms" (2nd Edition), 2002

Ill-Conditioned Problems

It is characteristic of ill-conditioned sets of equations that small percentage errors in the coefficients given may lead to large percentage errors in the solution



Alan Turing, Rounding-off errors in matrix process, 1948

- The condition number of linear system

$$\kappa_p(A) := \|A\|_p \, \|A^{-1}\|_p$$

- Usually we use the spectral norm (often omit $p = 2$)

$$\|A\|_2 := \left(\rho(A^T A)\right)^{\frac{1}{2}} \quad \Rightarrow \quad \kappa(A) = \kappa_2(A)$$

- Other useful condition-like terms:

$$\mathrm{cond}(A) := \||A^{-1}||A|\|_\infty$$

$$\mathrm{cond}(A, x) := \frac{\||A^{-1}||A||x|\|_\infty}{\|x\|_\infty}$$

$$\mathrm{cond}(A, x) \leqslant \mathrm{cond}(A) \leqslant \kappa_\infty(A)$$

Condition Number

- Condition number or sensitivity of a function at x: How sensitive w.r.t. perturbations?

$$\delta f(x) := f(x) - \hat{f}(x) \quad \Longrightarrow \quad \lim_{\epsilon \to 0} \sup_{\|\delta x\| \le \epsilon} \frac{\|\delta f(x)\|}{\|\delta x\|} \cdot \frac{\|x\|}{\|f(x)\|}$$

- Condition number of a matrix (linear operator)? Consider solving a linear system:

$$\max_{b, \delta b \neq 0} \frac{\|A^{-1}\delta b\|}{\|\delta b\|} \cdot \frac{\|b\|}{\|A^{-1}b\|} = \max_{\delta b \neq 0} \frac{\|A^{-1}\delta b\|}{\|\delta b\|} \max_{b \neq 0} \frac{\|b\|}{\|A^{-1}b\|}$$

$$= \max_{\delta b \neq 0} \frac{\|A^{-1}\delta b\|}{\|\delta b\|} \max_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \|A^{-1}\|\|A\|$$

Normwise Condition Number

- Condition number is crucial to the accuracy of fixed-precision direct solvers
- Moreover, it is critical for the convergence rate of iterative solvers

Error Analysis of Direct Solvers

- Fixed-precision computation results in inaccurate solutions ➔ Pivoting is crucial!
- Machine epsilon: the smallest possible real number such that $1.0 + \varepsilon > 1.0$

$$\varepsilon_{\text{double}} := 2^{-52} \approx 2.2204 \times 10^{-16}, \quad \varepsilon_{\text{single}} := 2^{-23} \approx 1.1921 \times 10^{-7}$$

Question: How big is the unit round-off in these cases? (precision $u = \varepsilon/2$)

- Perturbation analysis (backward error) for the LU decomposition

$$(A + \delta A)\hat{x} = b, \quad |\delta A| \le \gamma_{3n} |\hat{L}||\hat{U}| \qquad \gamma_n := \frac{nu}{1 - nu} \approx n\,\varepsilon$$

- Accuracy of direct solution methods (forward error) can be expected as, **roughly speaking**:

$$\frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} \le C_n \kappa_\infty(A)\varepsilon \qquad \boxed{\le \gamma_{3n}\, \kappa_\infty(A)\, \frac{\||\hat{L}||\hat{U}|\|_\infty}{\|A\|_\infty}}$$

Ref: N. J. Higham, "Accuracy and stability of numerical algorithms" (2nd Edition), 2002

Sparse Numerical Methods

Early history of sparse direct methods

Development and history of sparse direct methods

Iain S. Duff

iain.duff@stfc.ac.uk

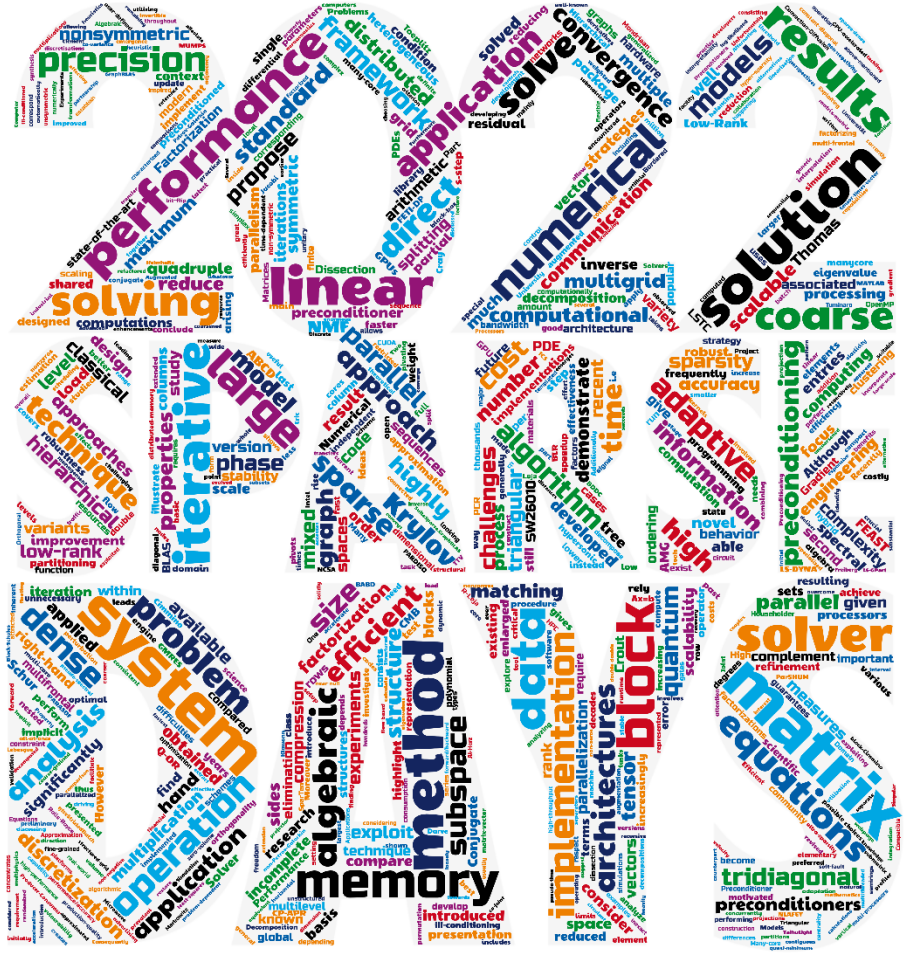
STFC Rutherford Appleton Laboratory

Oxfordshire, UK.

and

CERFACS, Toulouse, France

Homepage: http://www.numerical.rl.ac.uk/people/isd/isd.html



Source: https://archive.siam.org/meetings/la09/talks/duff.pdf

MUMPS

Sparse Days 2017 at CERFACS, Toulouse.
Celebrating the 70th birthday of Iain Duff.

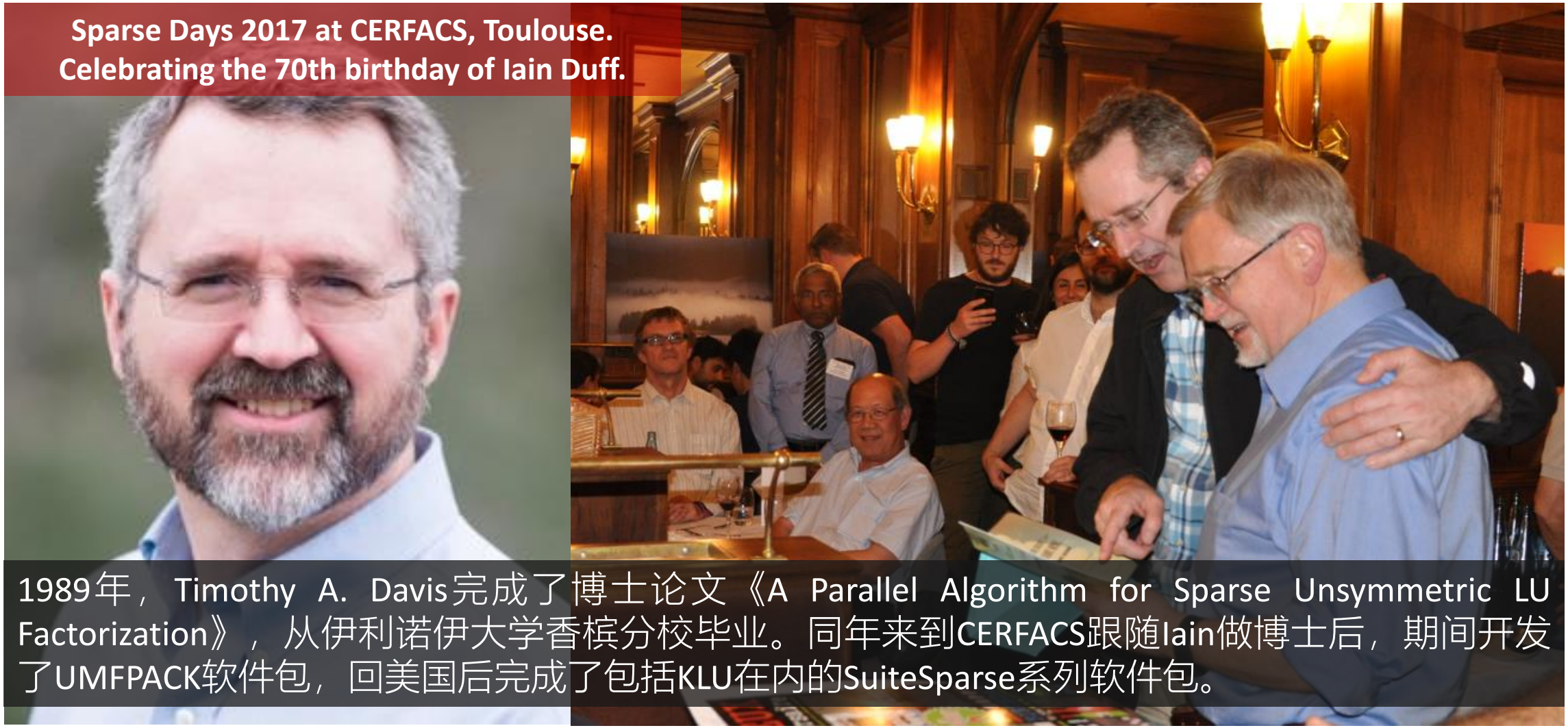
1972年，Iain Duff完成了博士论文《Analysis of Sparse Systems》，从牛津大学毕业；1986年开始在Harwell Laboratory领导Harwell Subroutine Library (from 1963, now HSL)中一系列程序的开发。

1990年，Iain的第一个博士生Patrick Amestoy实现了MUMPS库。完成了博士论文《Factorization of Large Sparse Matrices Based on a Multifrontal Approach in a Multiprocessor Environment》，从法国CERFACS毕业，后继续领导MUMPS软件包的开发。

Source: 中国石油大学（北京）刘伟峰2021年Solver21报告

UMFPACK/SuiteSparse

Sparse Days 2017 at CERFACS, Toulouse.
Celebrating the 70th birthday of Iain Duff.



1989年，Timothy A. Davis完成了博士论文《A Parallel Algorithm for Sparse Unsymmetric LU Factorization》，从伊利诺伊大学香槟分校毕业。同年来到CERFACS跟随Iain做博士后，期间开发了UMFPACK软件包，回美国后完成了包括KLU在内的SuiteSparse系列软件包。

Source: 中国石油大学（北京）刘伟峰2021年Solver21报告

SuperLU

1996年，Sherry Xiaoye Li完成了博士论文《Sparse Gaussian Elimination on High Performance Computers》，从加州大学伯克利分校毕业，后进入劳伦斯伯克利国家实验室开发了SuperLU系列软件包。

Source: 中国石油大学（北京）刘伟峰2021年Solver21报告

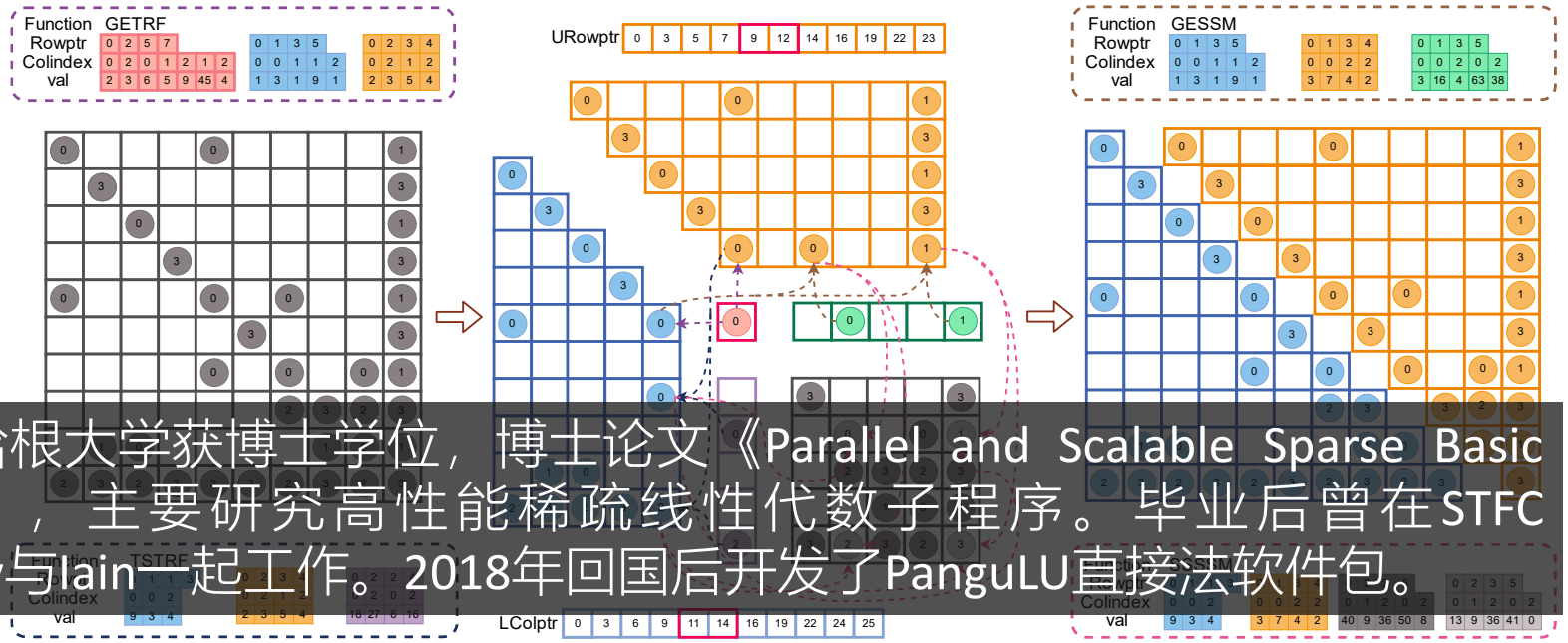
PARDISO

2000年，Olaf Schenk完成了博士论文《Scalable Parallel Sparse LU Factorization Methods on Shared Memory Multiprocessors》，从苏黎世联邦理工学院毕业，后在巴塞尔大学和卢加诺大学继续开发了PARDISO系列软件包。

Source: 中国石油大学（北京）刘伟峰2021年Solver21报告

PanguLU

面向异构分布式平台的开源直接法解法器（命名取“盘古开天地”之意， 经矩阵分解后，“天”为U矩阵，“地”为L矩阵）。该解法器使用C语言编写，使用MPI和CUDA完成并行架构。



2016年，刘伟峰于丹麦哥本哈根大学获博士学位，博士论文《Parallel and Scalable Sparse Basic Linear Algebra Subprograms》，主要研究高性能稀疏线性代数子程序。毕业后曾在STFC Rutherford Appleton Laboratory与Iain一起工作。2018年回国后开发了PanguLU直接法软件包。

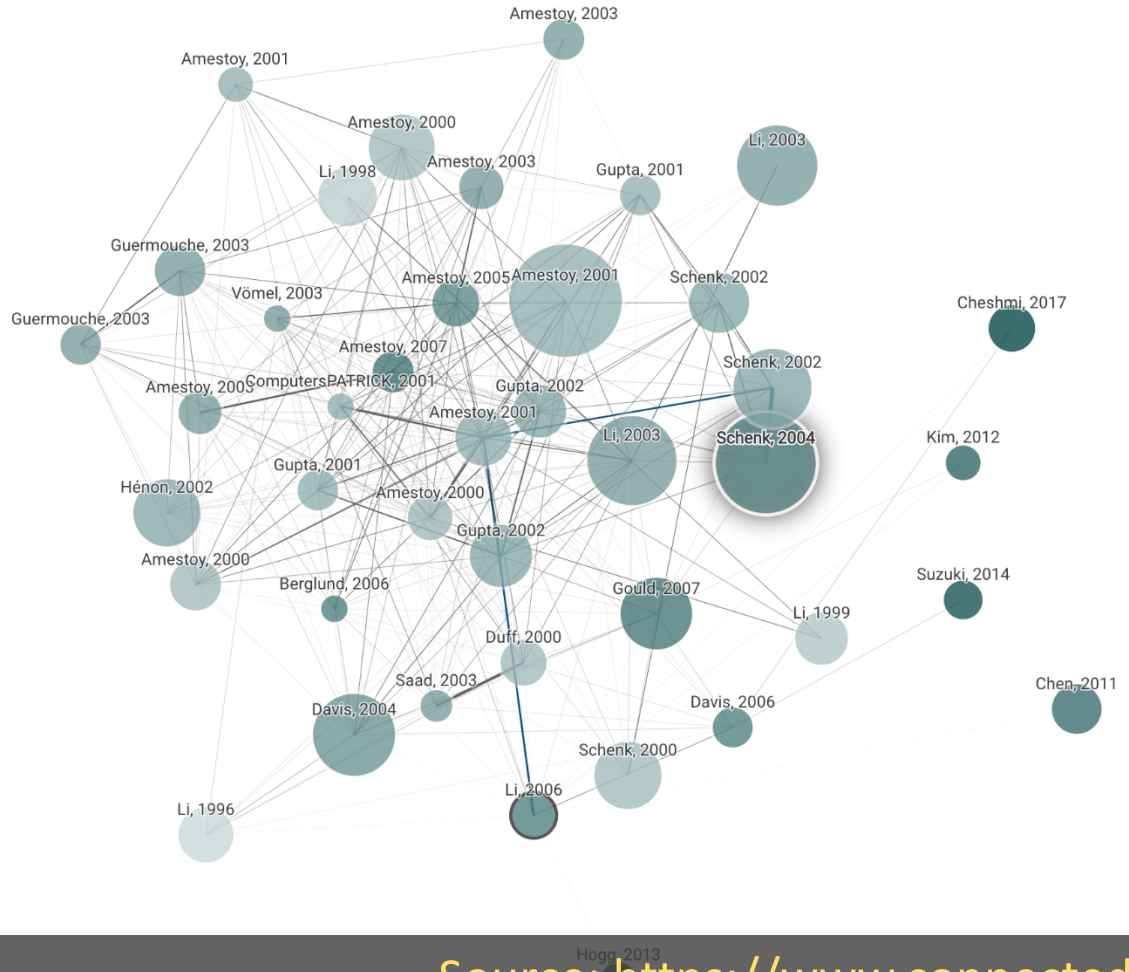
Direct Solver Packages

- **MUMPS:** http://mumps.enseeiht.fr/
- **SuiteSparse (CHOLMOD/UMFPACK/KLU):** https://people.engr.tamu.edu/davis/suitesparse.html
- **SuperLU:** https://portal.nersc.gov/project/sparse/superlu/
- **PARDISO:** https://www.pardiso-project.org/
- **SPOOLES:** http://www.netlib.org/linalg/spooles/spooles.2.2.html
- **WSMP:** https://researcher.watson.ibm.com/researcher/view_group.php?id=1426
- **H2Lib:** https://github.com/H2Lib/H2Lib
- **PanguLU:** https://gitee.com/ssslab/pangulu

List of Software for Linear Algebra by J. Dongarra and D. Sukkari

http://www.netlib.org/utk/people/JackDongarra/la-sw.html ➔ Redirected to Google Doc now

Direct Solver Papers



Solving unsymmetric sparse systems of linear equations with PARDISO

O. Schenk, K. Gärtner

2004, Future Gener. Comput. Syst.

1044 Citations

Save

Open in:

Supernode partitioning for unsymmetric matrices together with complete block diagonal supernode pivoting and asynchronous computation can achieve high gigaflop rates for parallel sparse LU factorization on shared memory parallel computers. The progress in weighted graph matching algorithms helps to extend these concepts further and unsymmetric prepermutation of rows is used to place large matrix entries on the diagonal. Complete block diagonal supernode pivoting allows dynamical interchanges of columns and rows during the factorization process. The level-3 BLAS efficiency is retained and an advanced two-level left-right looking scheduling scheme results in good speedup on SMP machines. These algorithms have been integrated into the recent unsymmetric version of the PARDISO solver. Experiments demonstrate that a wide set of unsymmetric linear systems can be solved and high performance is consistently achieved for large sparse unsymmetric matrices from real world applications.

Source: https://www.connectedpapers.com/

Iterative Solvers and Preconditioners

Iterative and preconditioning methods for linear systems

/03

Classical Iterative Methods

- A very long history: Newton, Euler, Guass, ...
- Gauss-Seidel, SOR, Krylov subspace methods, domain decomposition, multigrid, ...
- Steepest descent method, Newton's method, power method, ...
- Available for a large class of problems, cheap (in terms of computation and memory) to apply

$$\min_{x\in V} f(x) := \frac{1}{2}(Ax,x) - (b,x)$$

SPD

$$x^{\text{new}} = x^{\text{old}} + \alpha\, d$$

Richardson method

Optimal stepsize

Steepest Descent Method

$$\alpha_{\text{opt}} = \frac{(b - Ax^{\text{old}}, d)}{(Ad,d)} = \frac{(r^{\text{old}}, d)}{(Ad,d)}, \quad \text{with } r^{\text{old}} := b - Ax^{\text{old}}$$

$$d_{\text{steep}} := -\nabla f(x^{\text{old}}) = r^{\text{old}}$$

General Iterative Linear Solvers

Algorithm 1: Iterative solver

```
1 %% Given a nonsingular matrix A ∈ ℝ^{n×n}, b ∈ ℝ^n, and an initial guess x_0 ∈ ℝ^n;
2 for i = 0 : MaxIter or converged
3     Compute r_i ← b − A x_i;
4     Solve the error equation A e_i = r_i approximately;
5     Update x_{i+1} ← x_i + e_i;
6 end
```

- How to perform Line 4? Still the same problem!
- Simple approximations: Jacobi iteration, Gauss-Seidel, SOR, …
- Linear stationary methods
- Iterative refinement (IR) methods (Lecture 5)
- Line 4 alone ➔ Preconditioner

Conjugate Gradient Method

Algorithm 2: Conjugate gradient method

```
1  %% Given an initial guess u and a tolerance ε;
2  r ← f − Au,  p ← r;
3  while ‖r‖ > ε
4        α ← (r, r)/(Ap, p);
5        ũ ← u + αp;
6        r̃ ← r − αAp;
7        β ← (r̃, r̃)/(r, r);
8        p̃ ← r̃ + βp;
9        Update: u ← ũ,  r ← r̃,  p ← p̃;
10 end
```

Lucky breakdown

Applicable to sparse systems that are too large for direct methods

The Krylov subspace methods: “Top Ten Algorithms of the Century”, Dongarra and Sullivan, Computing in Science and Engineering, 2000

Ref: Hestenes and Stiefel. "Methods of Conjugate Gradients for Solving Linear Systems". Journal of Research of the National Bureau of Standards 49, 1952

Preconditioning

- Convergence of the Richardson method

$$\left\|u-u^{(m)}\right\|_{\mathcal{A}} \leq \left(\frac{\kappa(\mathcal{A})-1}{\kappa(\mathcal{A})+1}\right)^{m} \left\|u-u^{(0)}\right\|_{\mathcal{A}}.$$

- Convergence of the conjugate gradient method

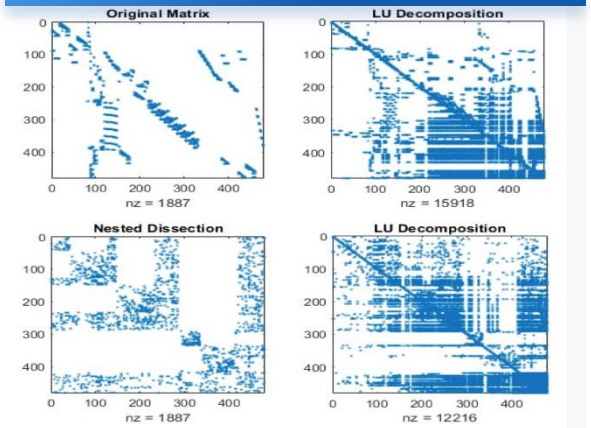
$$\left\|u-u^{(m)}\right\|_{\mathcal{A}} \leq 2\left(\frac{\sqrt{\kappa(\mathcal{A})}-1}{\sqrt{\kappa(\mathcal{A})}+1}\right)^{m} \left\|u-u^{(0)}\right\|_{\mathcal{A}}.$$

These classical results are only upper bounds. And they are usually pessimistic!

- Q: How accurate are these estimates (upper bounds)? Distribution of eigenvalues!
- Reduce condition number: Meijerink, van der Vorst 1977 (IC preconditioner) ...
- How to precondition? Diagonal, SGS, SSOR, ILU, Sparse Approximate Inverse, ...
- Domain decomposition, multigrid, nonlinear preconditioning, ...
- Problem dependent, usually requires at least the coefficient matrix

Popular Preconditioning Techniques

基于代数的方法

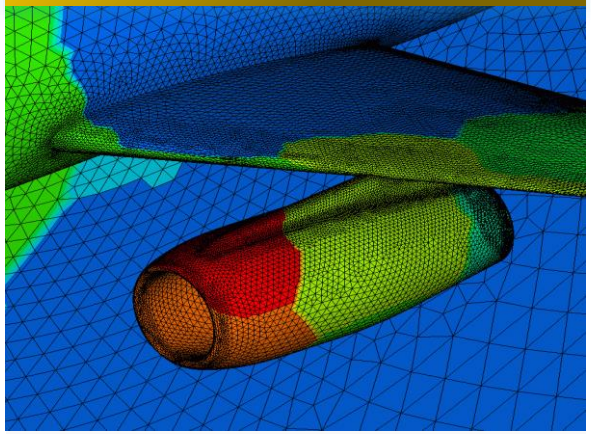


01.

LU, ILU, SAI, …

纯代数，通用性强，稳健性高，用户友好；效率一般不高，并行可扩展性较差。

基于区域分解的方法

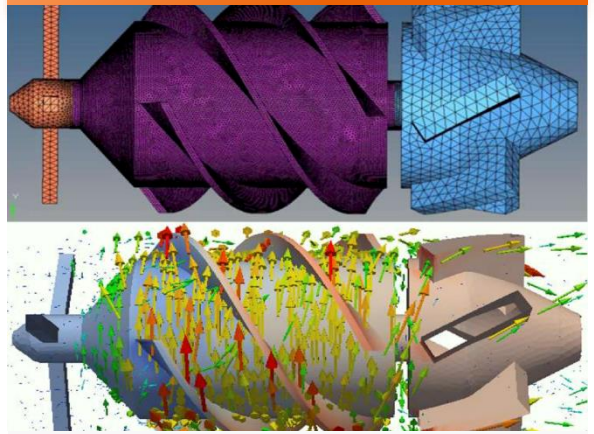


02.

DDM, RAS, FETI-DP, …

可以基于网格进行，效率高，通用性较强，可扩展性较强；难以兼顾通用性与最优性。

基于物理的方法



03.

Block Preconditioners

算法灵活，基于成熟算法开发，效率高，可扩展性强；通用性弱，用户友好度差。

Method of Subspace Corrections

- 利用子空间校正框架来介绍Schwarz方法：求解 $u \in V$，使 $Au = f$

Ref: J. Xu, SIAM Review 1992

- 构造一个合适的空间分解：

$$V = \sum_{i=1}^{J} V_i, \quad u = \sum_{i=1}^{J} u_i = \sum_{i=1}^{J} I_i u_i = \Pi \mathbf{u}$$

Divide-and-Conquer

其中 $\Pi := (I_1, \ldots, I_J), \quad \mathbf{u} := (u_1, \ldots, u_J)^T$

- 子空间求解器： S_i： $V_i' \to V_i$

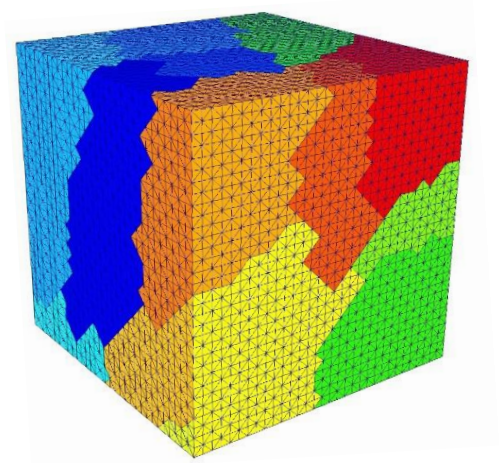
$$(A_i u_i, v_i) := (A u_i, v_i), \ u_i, v_i \in V_i, \quad S_i \approx A_i^{-1}$$

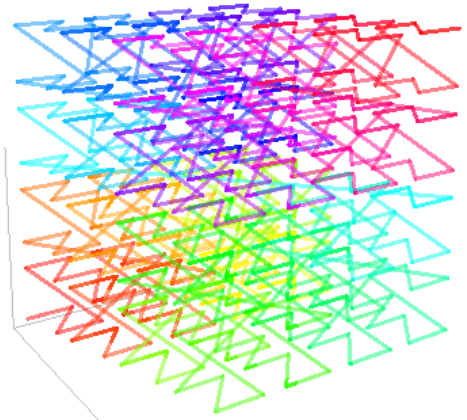
Solve small problems

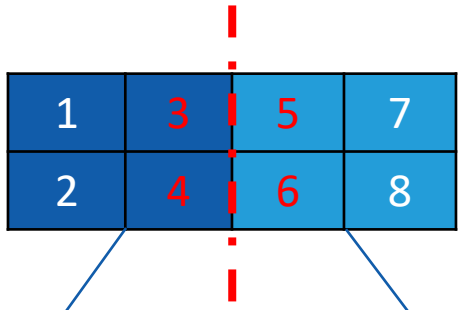
- 加性子空间校正PSC（并行方法，并行程度高）： $u \leftarrow u + B(f - Au), \quad B := \sum_{i=1}^{J} I_i S_i I_i^T$

- 乘性子空间校正SSC（串行方法，收敛速度较快）： $u \leftarrow u + S_i I_i^T (f - Au), \ i = 1 : J$

Domain Decomposition Methods



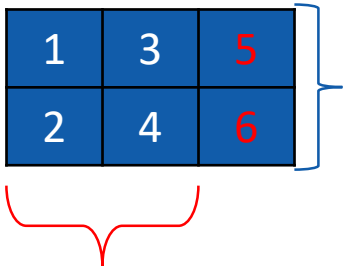




网格划分

数据分发

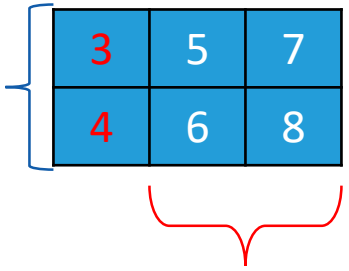
进程1



局部网格

Ghost区
需要进行
数据传递

进程2



局部计算

局部网格

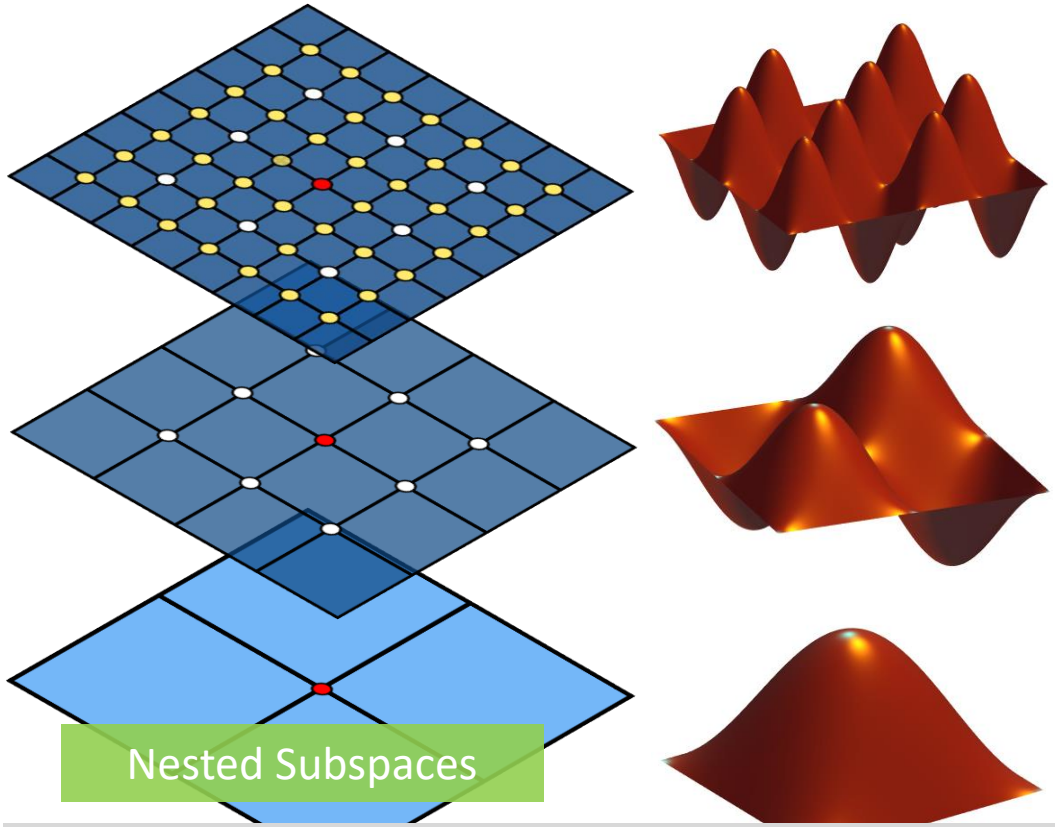
多目标优化问题：每个部分的工作量的变化极小，界面面积极小，裂缝等特征不跨区域……

Divide-and-Conquer

初始网格处理 → 空间填充曲线 → 负载平衡剖分 → 子区域求解

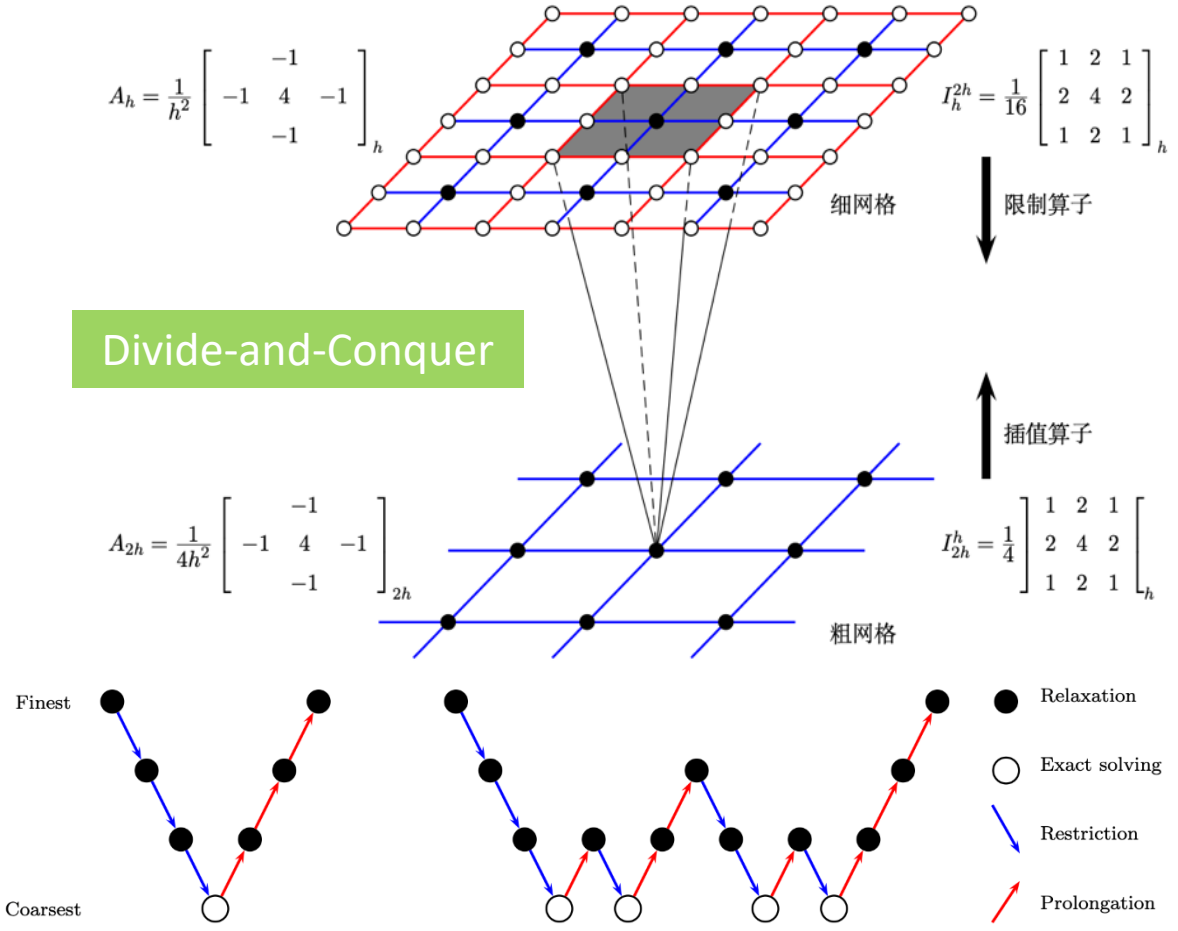
Geometric Multigrid Methods

Q: How to handle global low-frequency error components (small eigen components)?



Nested Subspaces

Source: 吴淑红, 张晨松, 王宝华, 冯春生, 许进超
油藏数值模拟中的线性代数求解方法, 2020



Multigrid V-Cycle Method

Algorithm 3: V (multigrid V-cycle)

Recursive Version

```
1  %% For any level ℓ, give a nonsingular matrix A_ℓ and prolongation P_ℓ;
2  %% Compute the residual r_ℓ of the current iteration;
3  x_ℓ ← M r_ℓ; %% pre-smoothing
4  if ℓ > 0 %% not the coarsest level yet
5      r_ℓ ← r_ℓ − A x_ℓ; %% form residual
6      r_{ℓ−1} ← P_ℓ^T r_ℓ; %% restriction
7      e_{ℓ−1} ← V(A_{ℓ−1}, r_{ℓ−1}, P_{ℓ−1}, ℓ − 1); %% recursive call
8      e_ℓ ← P_ℓ e_{ℓ−1}; %% prolongation
9      x_ℓ ← x_ℓ + e_ℓ; %% correction
10     r_ℓ ← r_ℓ + A e_ℓ; %% update residual (optional)
11     x_ℓ ← x_ℓ + M^T r_ℓ; %% post-smoothing (optional)
12 end
13 return x_ℓ
```

Efficiency vs User-Friendliness

问题规模	网格剖分	64x64x64			128x128x128	256x256x256	512x512x512
	变量个数	274,625			2,146,689	16,974,593	135,005,697
稀疏直接法软件 Intel MKL Pardiso 北京超级云超算	计算核数	8x1	16x1	32x1	16x8	16x64	16x512
	求解时间	**5.38s**	**3.86s**	**3.26s**	**59.78s**	**999.46s**	内存不足
几何多重网格法 FASP求解器软件 个人笔记本电脑	计算核数	1x1			1x1	1x1	1x1
	求解时间	**0.030s**			**0.303s**	**2.815s**	**23.54s**

三维Poisson方程（均匀网格七点差分格式）的线性解法器对比，稀疏直接法Pardiso（北京超级云超算）和几何多重网格法FASP（笔记本电脑）：线程数x进程数。2020年，北京超级云计算中心A分区以Linpack测试性能3.74PFlops，获中国HPC TOP100榜单第三名及通用CPU算力第一名。单节点配两块AMD EPYC7452共64核心256GB内存。

Importance of Preconditioning

中国石油勘探开发研究院：个性化井网设计软件求解性能测试与优化

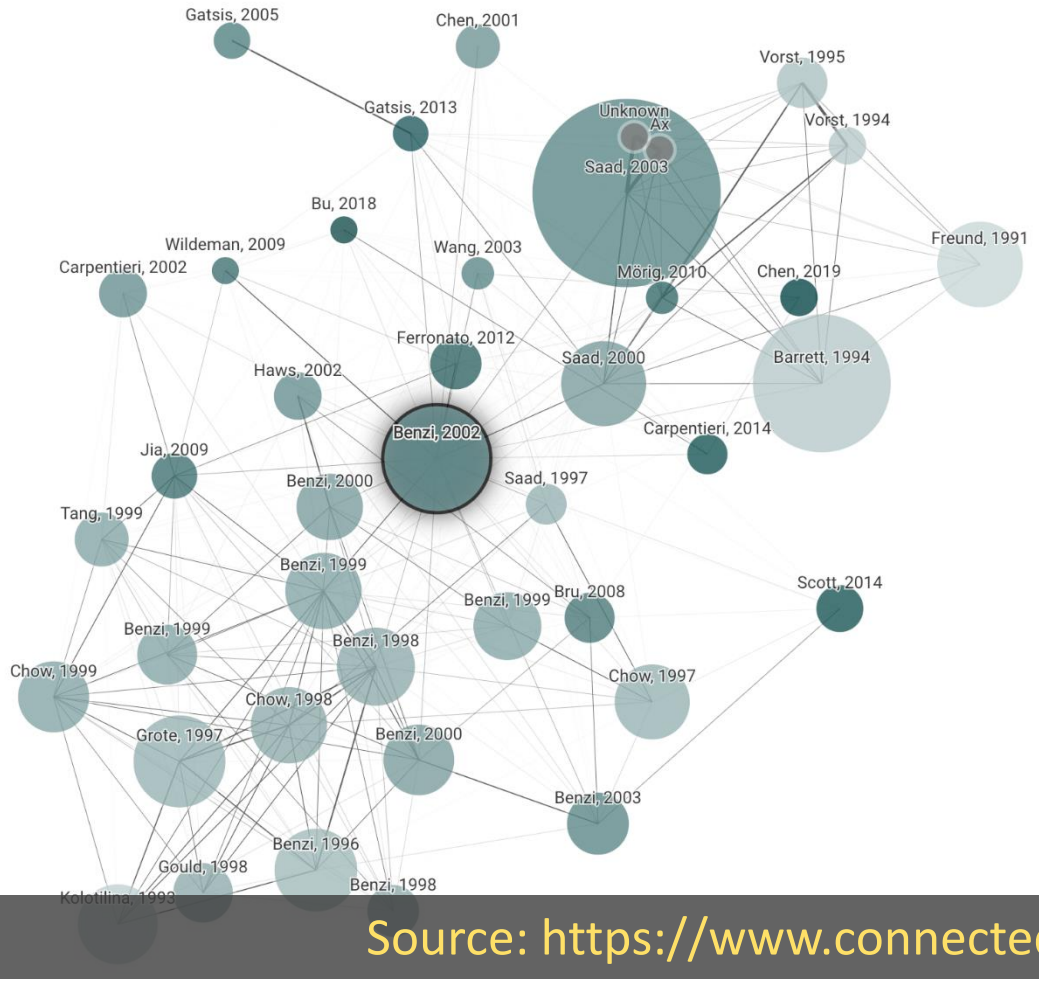
- 通过斯坦福大学的Industrial Consortium获得油藏模拟器全套代码
- 18万行代码，消化这些代码比较困难
- 一些算例不能算，一些算法不能算或者算得很慢……

非程序错误

AMG求解时间	算例A			算例B		
	耗时（秒）	调用次数	加速比	耗时（秒）	调用次数	加速比
修改前	1705.188	192	---	2034.765	107	---
修改后	7.815	72	219倍	94.475	37	22倍

预条件方法代码修改（其中的一行）前后计算速度对比

Preconditioning Papers



Preconditioning techniques for large linear systems: a survey

M. Benzi

2002

1155 Citations

Save

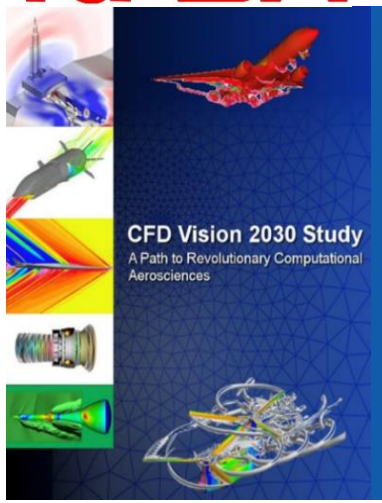
Open in:

This article surveys preconditioning techniques for the iterative solution of large linear systems, with a focus on algebraic methods suitable for general sparse matrices. Covered topics include progress in incomplete factorization methods, sparse approximate inverses, reorderings, parallelization issues, and block and multilevel extensions. Some of the challenges ahead are also discussed. An extensive bibliography completes the paper.

Source: https://www.connectedpapers.com/

Solver is Crucial to Scalability

NASA



- 大规模、高精度的模拟需要变革性的算法支撑
- 大规模网格并行生成和自适应仍是主要技术瓶颈
- 分析和优化过程需要稳健的自动化的求解器技术
- 精细模拟带来大量的数据需要存储、处理和传输
- HPC硬件发展迅速，其发展趋势难以预测

精细模拟 → 并行计算 → 隐式方法 → 线性解法 → 迭代方法 → 可靠解法 → 容错方法 → 稳健解法

- 快速线性求解器是大规模计算中的主要瓶颈，形成**高效、通用、可扩展**的线性求解器软件是一个难题
- 快速线性求解器是大规模数值模拟的重要组件，在一些应用问题的数值模拟中占用了80%以上的时间
- 对超算硬件的性能进行排名常采用线性求解器作为标准测试（如HPL、HPCG等）

Combining Direct and Iterative Methods

- Multigrid methods: Using direct method as coarse grid solver
- Domain decomposition methods: Using direct method on local subdomains and “direct” preconditioner on interfaces
- Partial factorization: Incomplete factorizations as preconditioners (like ILU methods)
- Direct method used in part of a preconditioner (like MSP for the well part)
- Block iterative methods: Using direct method on sub-blocks (like Schwarz preconditioners)
- Low-precision direct method as a preconditioner (like GMRES-IR)
- Iterative refinement: improve precision of direct solvers (like LU-IR)
- Factorization of nearby problem as a preconditioner

Iain Duff. The SIAM Conference on Applied Linear Algebra. October 26-29, 2009. Monterey, California.

Reading and Thinking

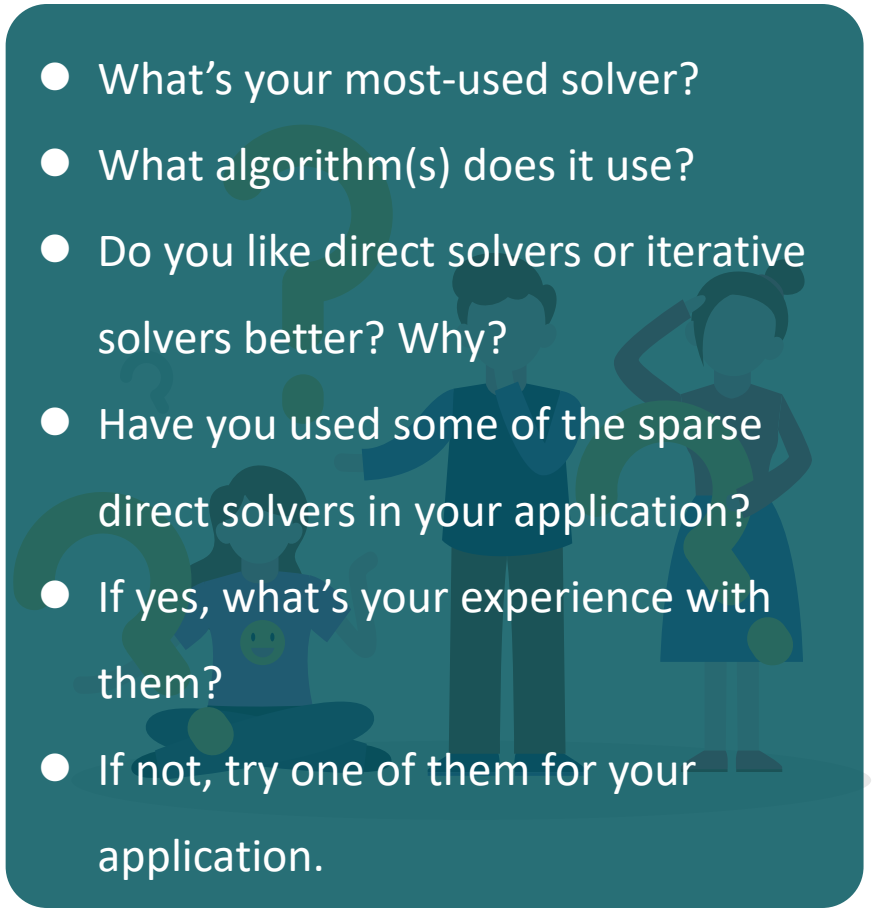
Acta Numerica (2016), pp. 383–566
doi:10.1017/S0962492916000076

A survey of direct methods for sparse linear systems

Timothy A. Davis
Department of Computer Science & Engineering,
Texas A&M University,
College Station, TX 77843-3112, USA
E-mail: davis@tamu.edu

Sivasankaran Rajamanickam
Center for Computing Research,
Sandia National Laboratories,
Albuquerque, NM 87185-1320, USA
E-mail: srajama@sandia.gov

Wissam M. Sid-Lakhdar
Department of Computer Science & Engineering,
Texas A&M University,
College Station, TX 77843-3112, USA
E-mail: wissam@tamu.edu



- What's your most-used solver?
- What algorithm(s) does it use?
- Do you like direct solvers or iterative solvers better? Why?
- Have you used some of the sparse direct solvers in your application?
- If yes, what's your experience with them?
- If not, try one of them for your application.

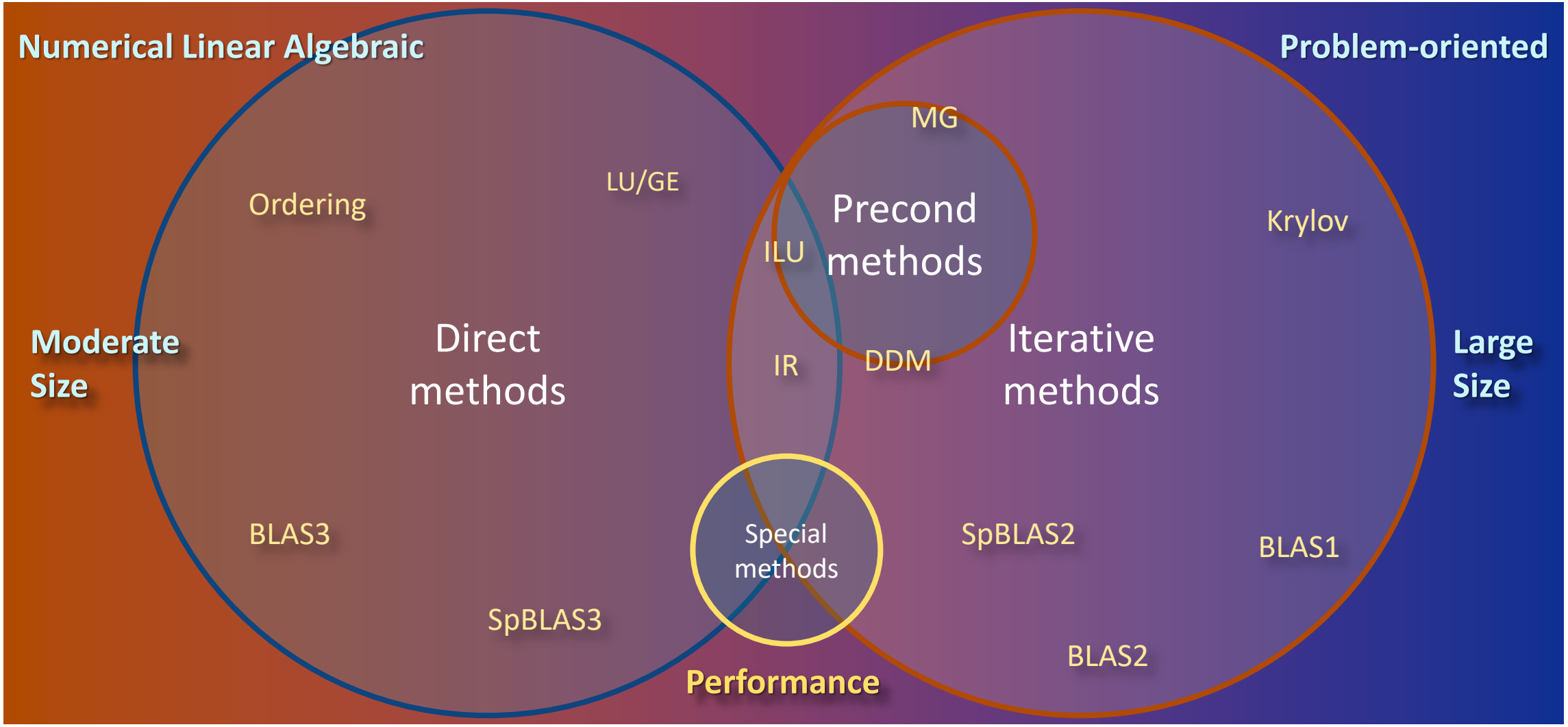
Contact Me

- Office hours: Mon 14:00—15:00
- Walk-in or online with appointment
- zhangcs@lsec.cc.ac.cn
- http://lsec.cc.ac.cn/~zhangcs

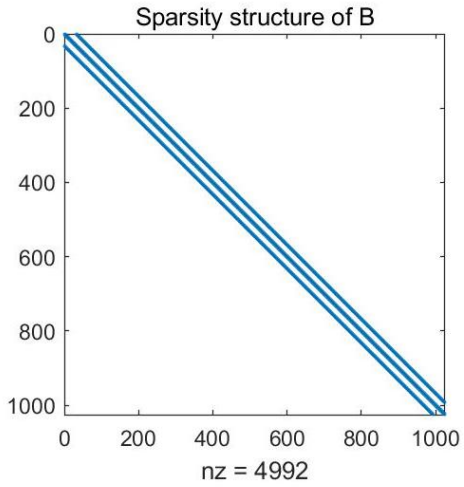
My sincere gratitude to:

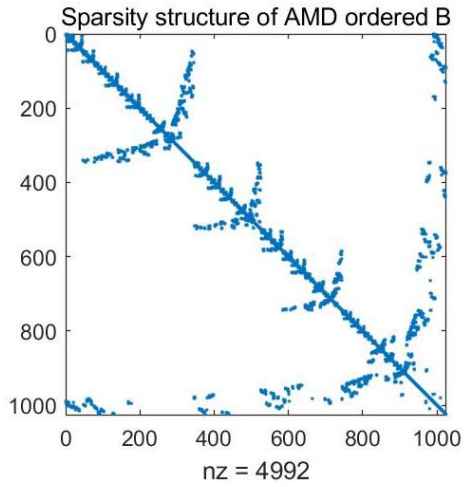
Weifeng Liu, Shihua Gong, Bin Dai, Shizhe Li, Jindong Wang

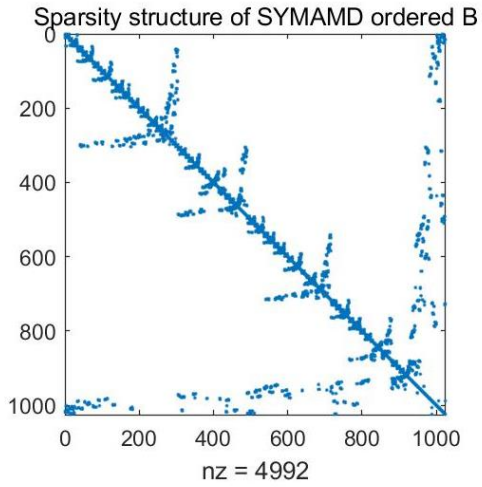
Review

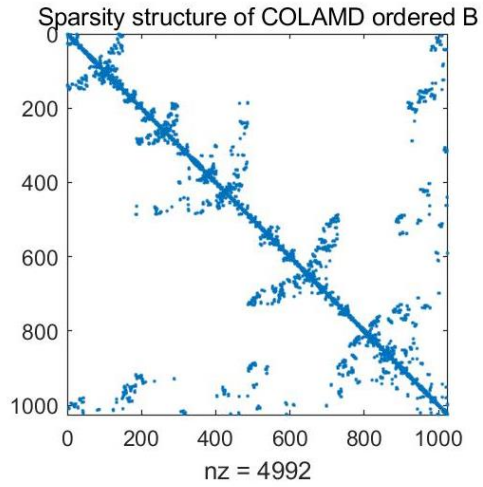

Numerical Linear Algebraic
Problem-oriented
MG
Ordering
LU/GE
Precond methods
ILU
Krylov
Moderate Size
Direct methods
IR
DDM
Iterative methods
Large Size
BLAS3
Special methods
SpBLAS2
BLAS1
SpBLAS3
BLAS2
Performance

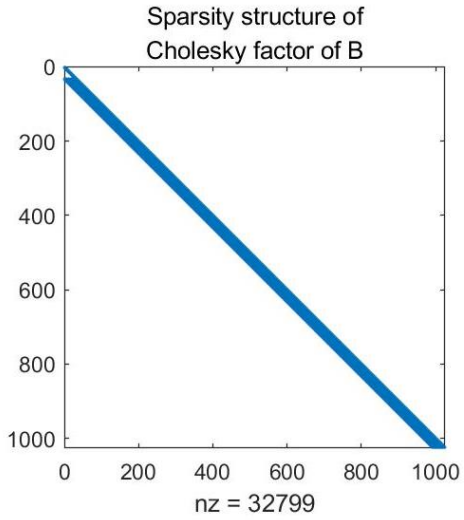
Various Reordering Schemes

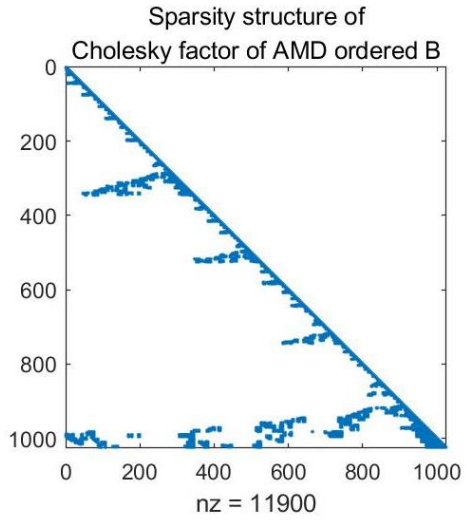


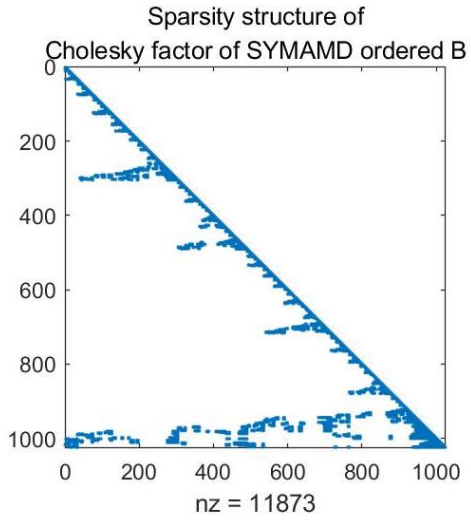


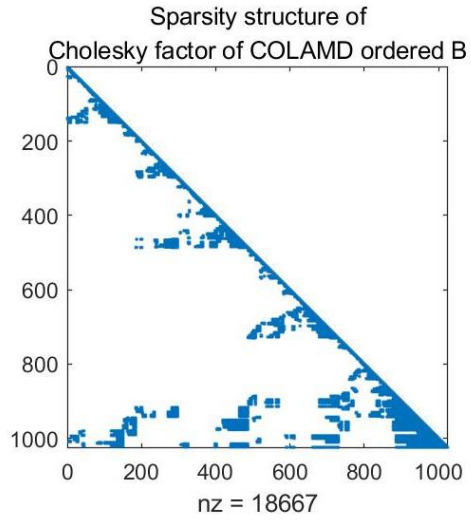




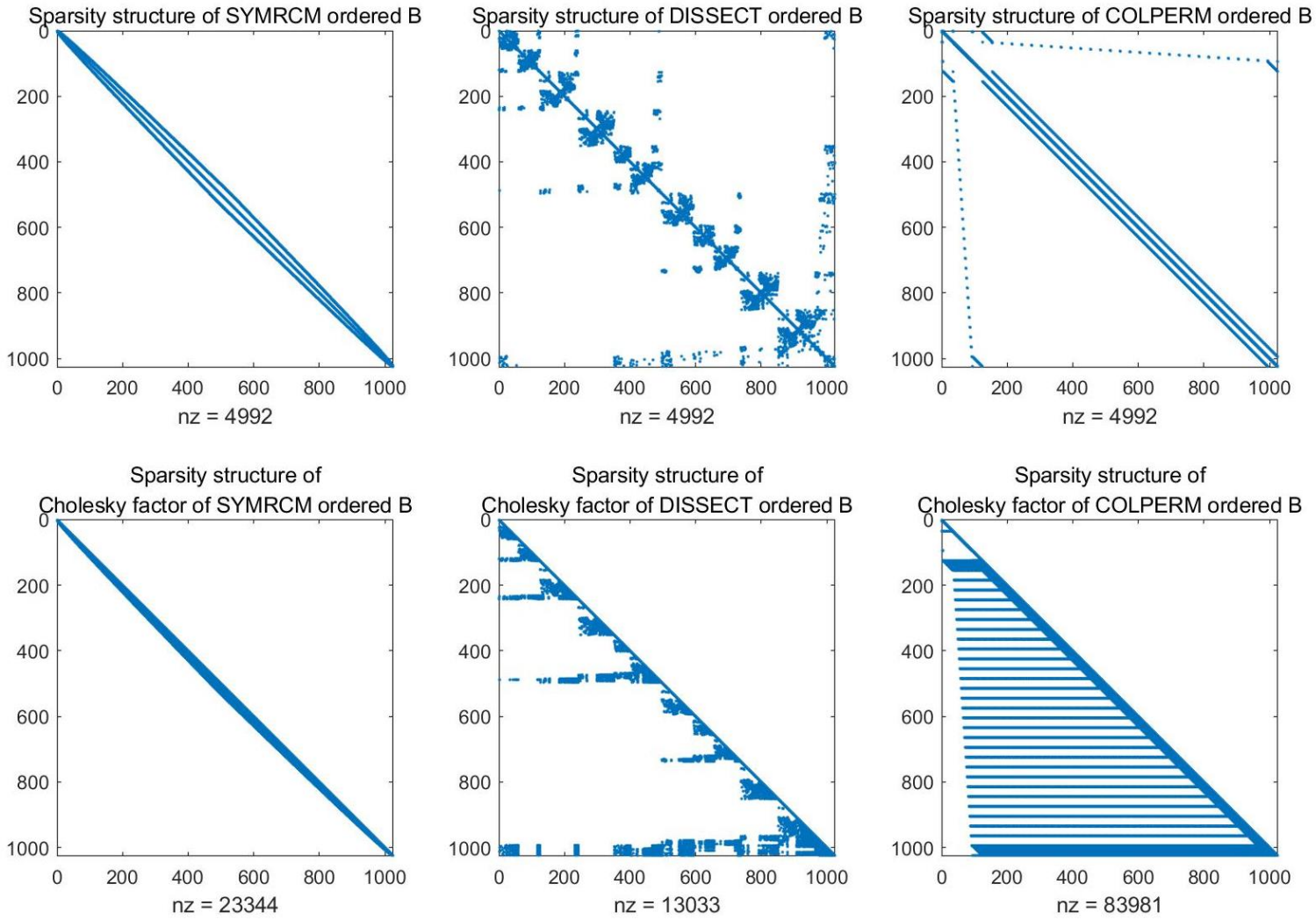








Various Reordering Schemes



Source: Tested by Bin Dai using Matlab with gallery test problems (poisson and neumann on 32×32 grid)

IC Preconditioning with Reordering

表 5: mesh size = 32×32; preconditioner: IC with droptol = 0.1

method	Order	size	Iteration	relative residual	number: Fill-ins
cg	original	1024	84	8.2e-07	\
pcg	original	1024	27	7.6e-07	4096
pcg	amd-ordered	1024	34	7.1e-06	5692
pcg	symamd-ordered	1024	34	7.3e-07	5676
pcg	colamd-ordered	1024	34	5.2e-07	4320
pcg	symrcm-ordered	1024	27	9.3e-07	4096
pcg	dissect-ordered	1024	37	8.1e-07	5434
pcg	colperm-ordered	1024	26	9.5e-07	4096

- Reordering might (or not) reduce number of fill-ins, but sometimes results in more iterations
- Q: Can reordering improve overall performance? It is not clear at all …

中国科学院大学
夏季强化课程
2022

Fast Solvers for Large Algebraic Systems

Chensong Zhang, AMSS

http://lsec.cc.ac.cn/~zhangcs

Release version 2022.06.22