2024年秋季·多层迭代法
课程内容简介

日期：2024.09.03

教师：张晨松，中国科学院数学与系统科学研究院

Contact

Information:

- https://lsec.cc.ac.cn/~zhangcs/include/mg2024.html
- Contact: zhangcs@lsec.cc.ac.cn
- Office hours: By appointment

My sincere gratitude to:

Shizhe Li, Bin Dai, Yan Xie, Li Zhao

Agenda

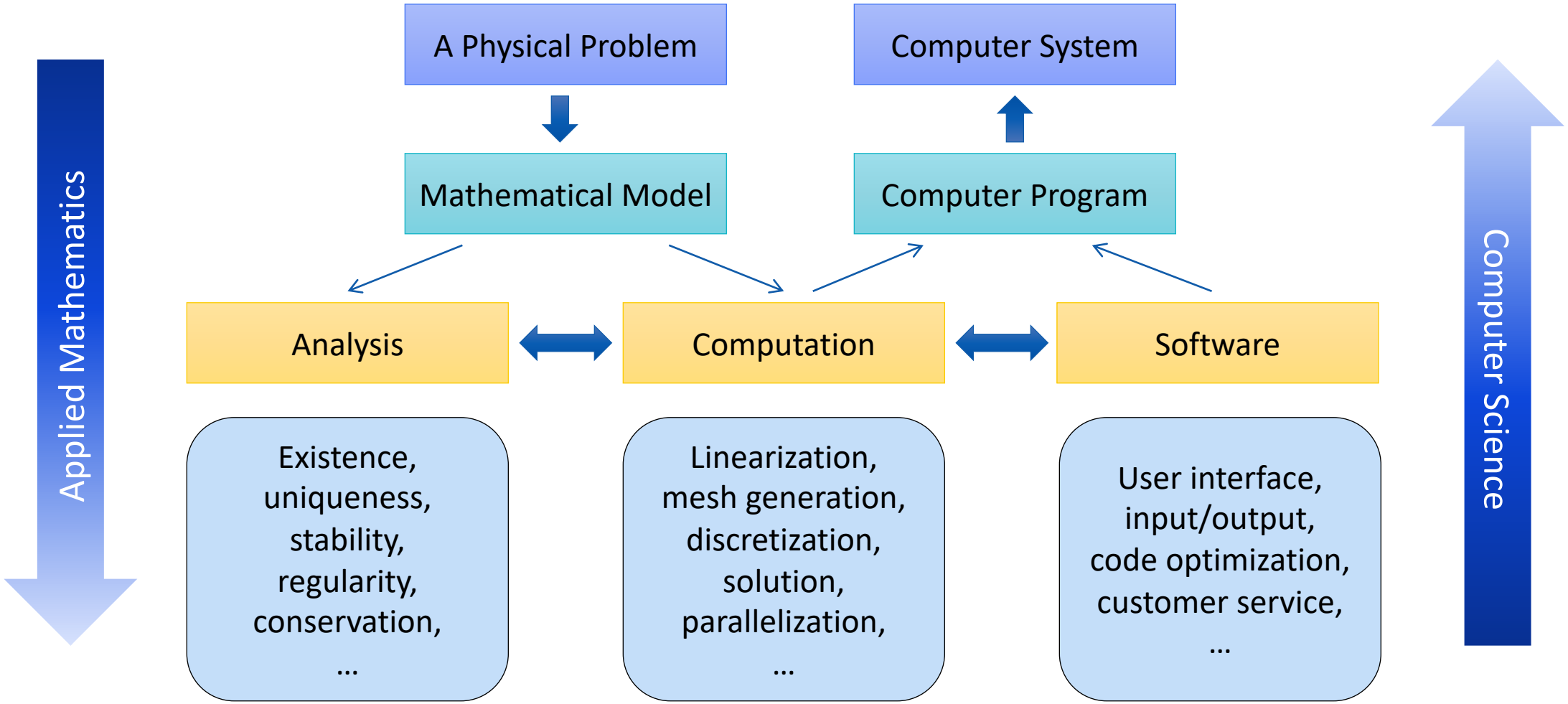
- Part 1: Introduction to linear solvers
- Part 2: Some classical iterative solvers
- Part 3: Multilevel iterative solvers
- Part 4: Theory based on subspace corrections
- Part 5: Some applications and their solvers

Grades

- Attendance: 5%
- Homework problems: 25%
- Final project: 70%
- Extra points: Find typos in lecture notes and slides

Section 1. Introduction to Linear Solvers

The Third Paradigm of Scientific Discovery

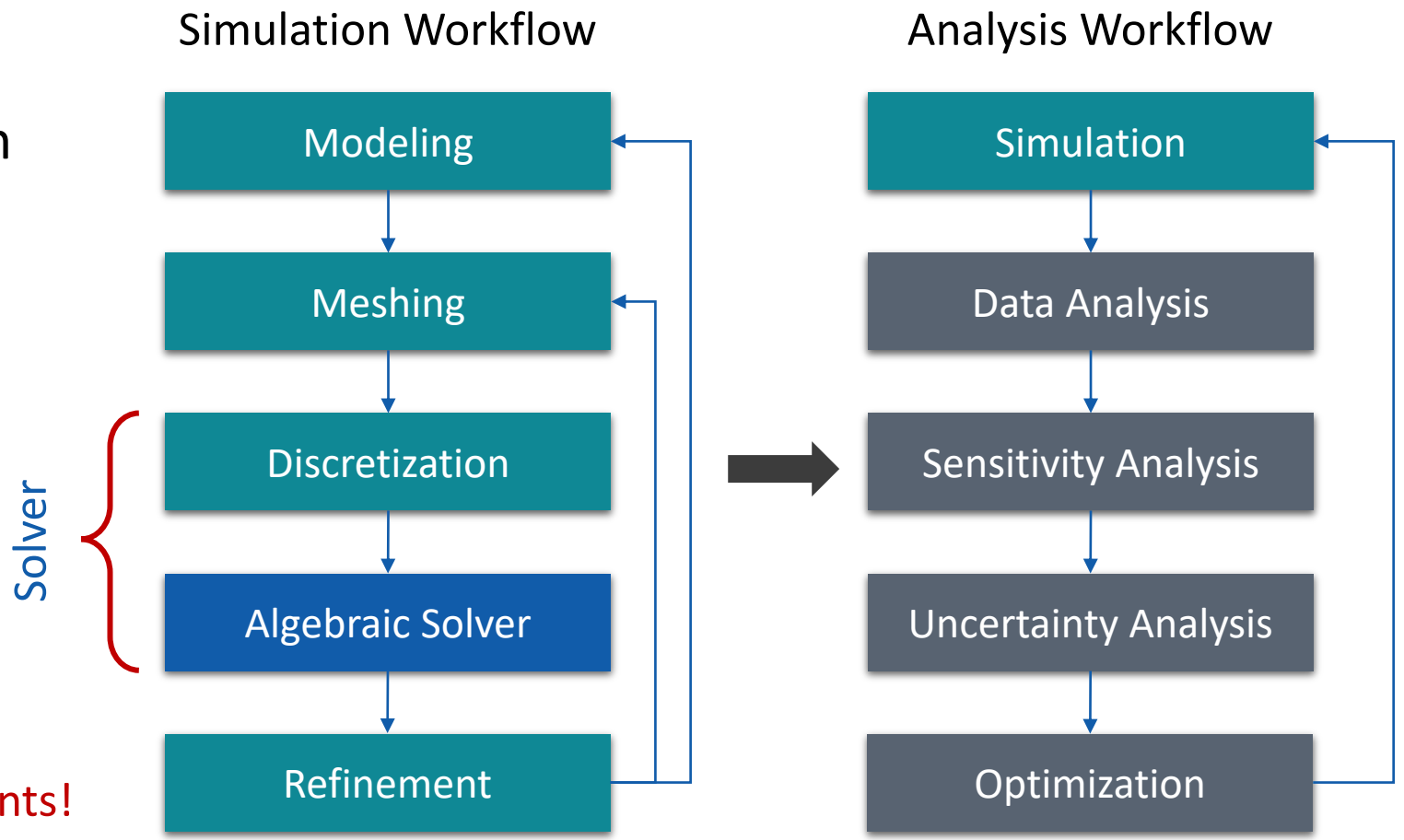


Why Simulation So Important

In many situations, we have very limited **theories** nor can we do **experiments**:

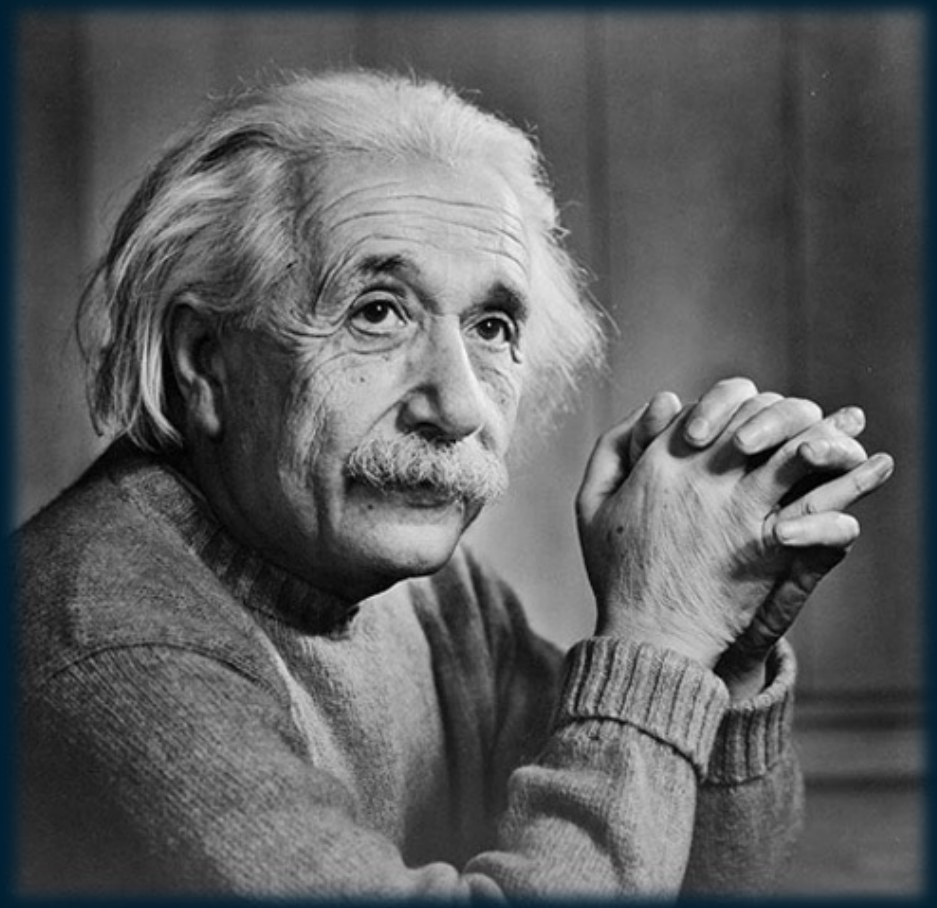
- Too slow
- Too difficult
- Too expensive
- Too dangerous

Simulation ≠ Numerical experiments!



In this lecture, Solver := Algebraic Solver (Solution Methods & Implementations)

Wisdoms in Modeling

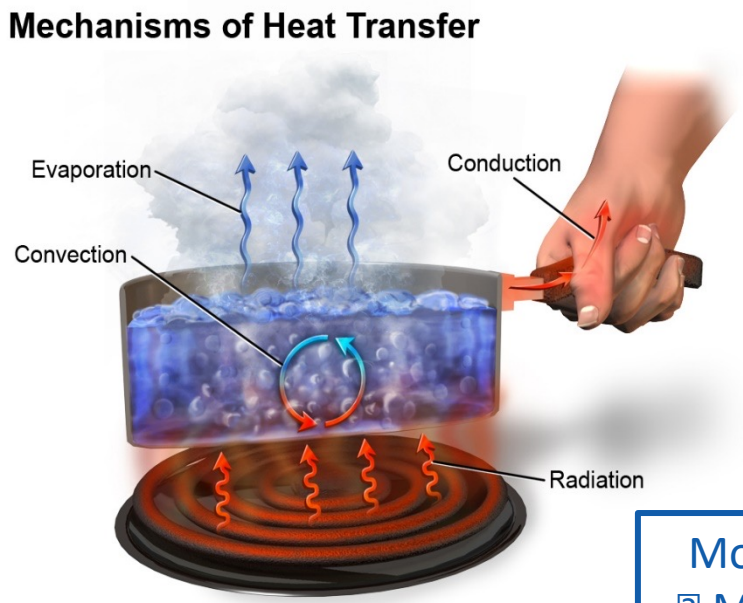


Make everything
as simple
as possible
but no simpler

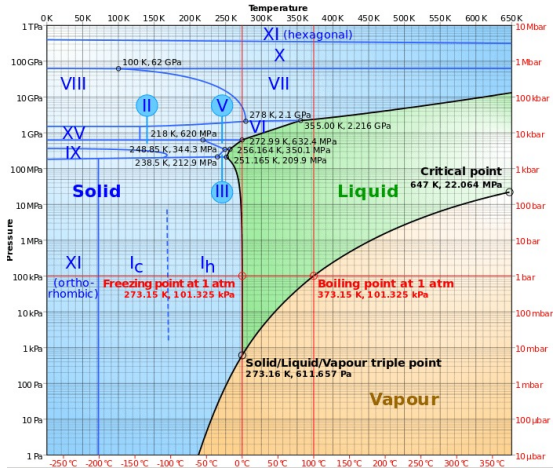
[Einstein]
attributed

'On the Method of Theoretical Physics', lecture delivered at Oxford, 10 June 1933

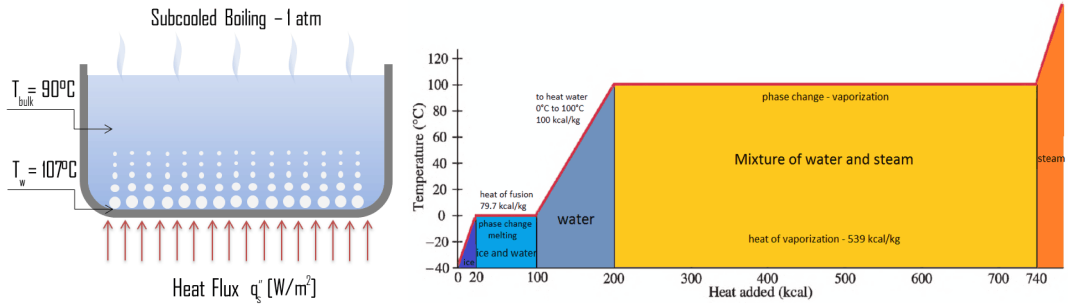
Boiling Water

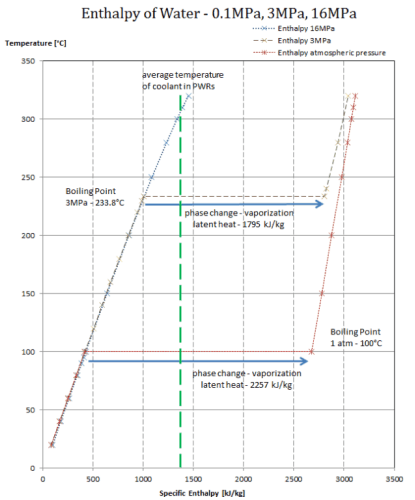


Source: https://thermtest.com/



Monophase
⍰ Multiphase





Source: https://www.thermal-engineering.org/

Wisdoms in Numerical Simulation

数值方法设计思路

原始模型

无穷维 | 非线性 | 全局 | 复杂 | 多变量

数值模型

有限维 | 线性 | 局部 | 简单 | 少变量

复杂模型的并行数值模拟

- 输入输出
- 初始化
- 耦合方程处理
- 时间离散（自适应时间步长）
- 空间离散（网格剖分、并行划分、自适应等）
- 线性化（Newton法、步长搜索、自适应）
- 线性求解器（解耦方法、预条件方法、迭代方法）
- 并行支撑技术（区域划分、并行通信）
- 辅助方程计算（非线性代数方程求解）

Linear Solution Methods

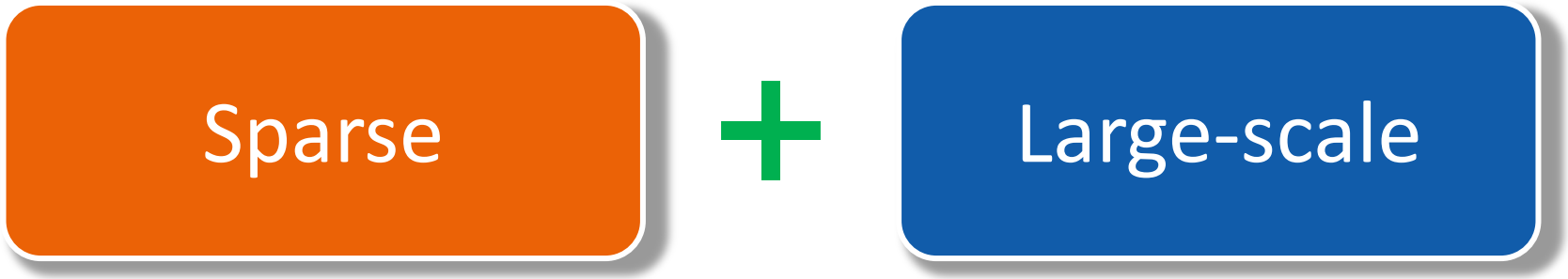
Given a large sparse matrix $A \in \mathbb{R}^{N\times N}$ and $\vec{f} \in \mathbb{R}^N$, find $\vec{u} \in \mathbb{R}^N$ such that $A\vec{u} = \vec{f}$

- It looks easy! Does it? It is mostly true 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
- Design goals of linear solvers:

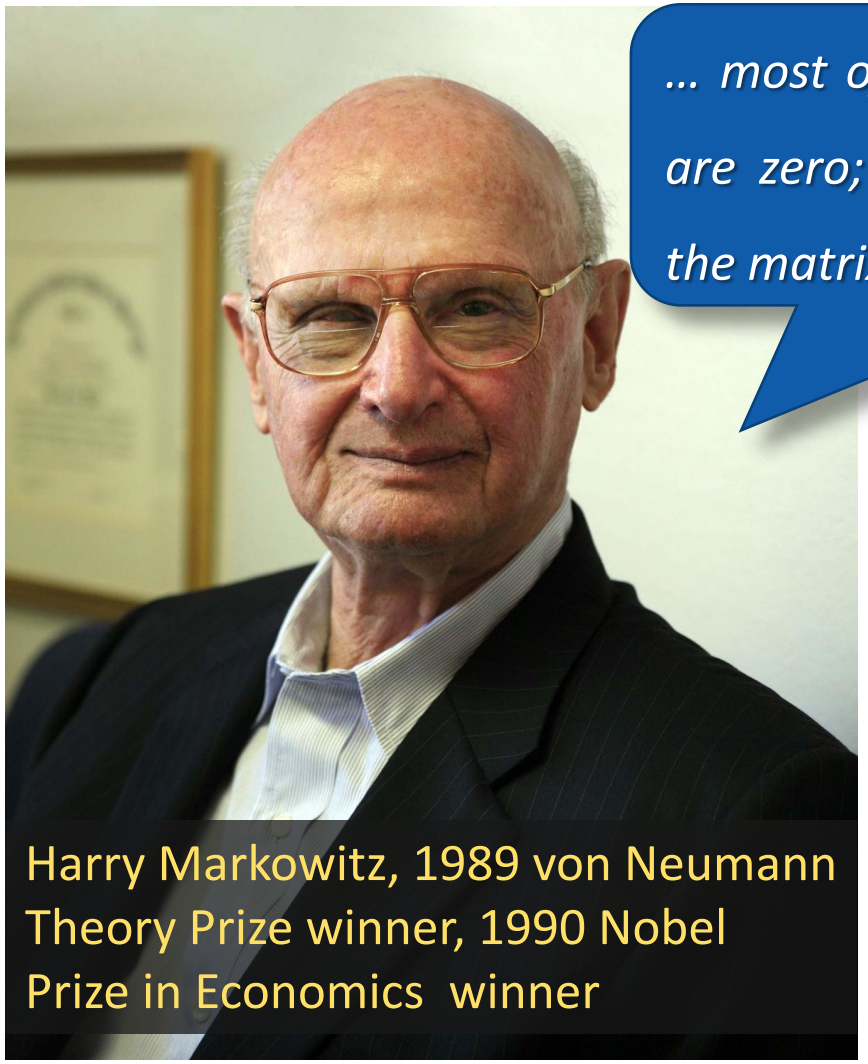
  accuracy, convergence, applicability, efficiency, optimality, user-friendliness, robustness, scalability, reliability, resilience, cost-effectiveness, ...

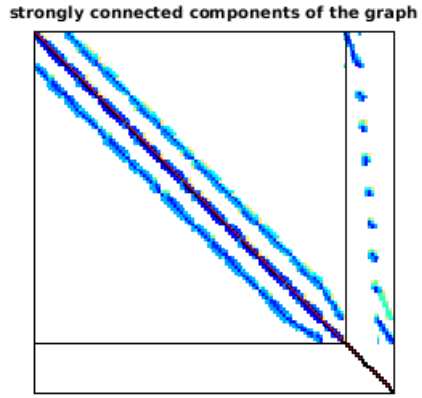
Section 2. Large-scale Linear Systems

Keywords



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? **Weifeng Liu's lectures**

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

Data Structures for Sparse Matrices

CSR

CSC

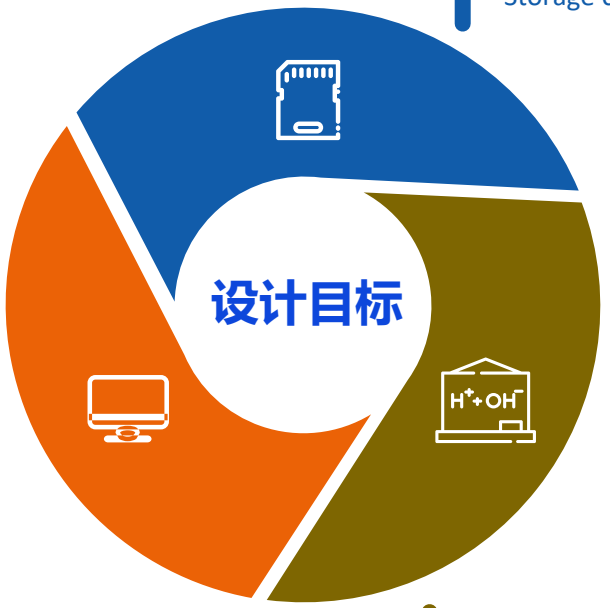
DIA

ELL

COO

HYB

CSR5



存储开销小

Storage cost

计算性能高

Computing efficiency

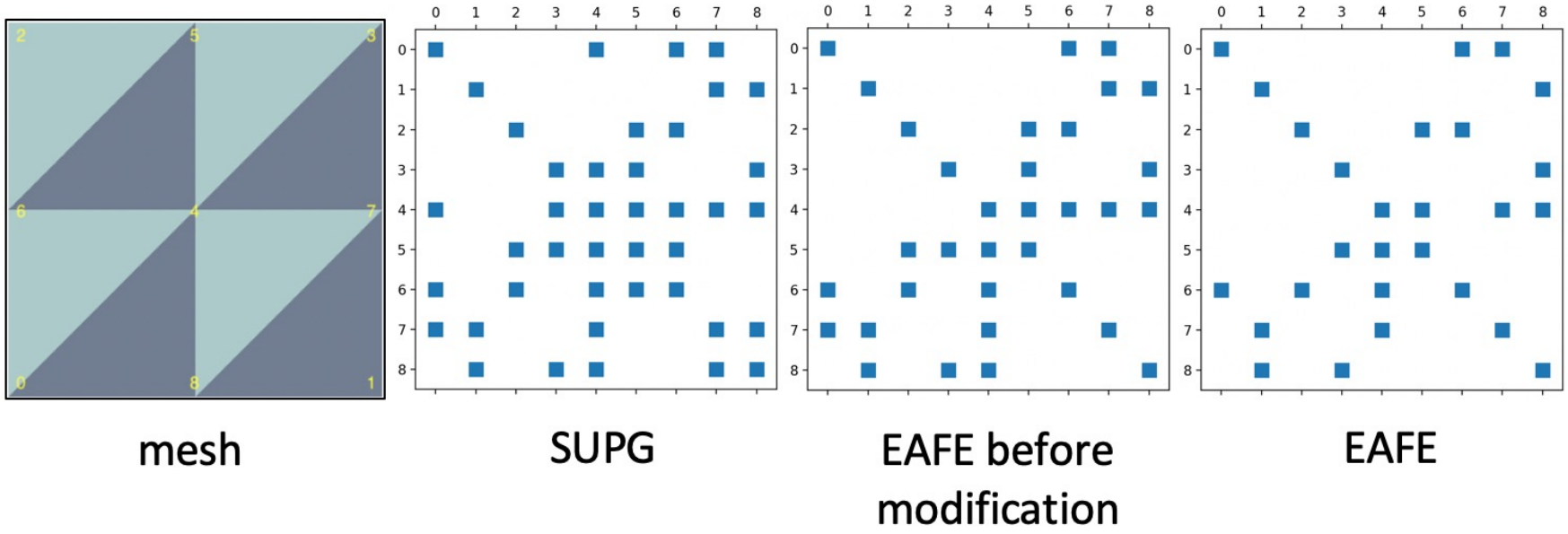
算法通用性强

Suitable for various algorithms

- 元素操作（读写某些元素）
- 矩阵操作（行列交换、转置）
- 矩阵运算（SpMV、SpGEMM）
- 迭代算法（Jacobi方法、GS方法）
- 代数算法（LU、ILU、SAI、AMG方法）

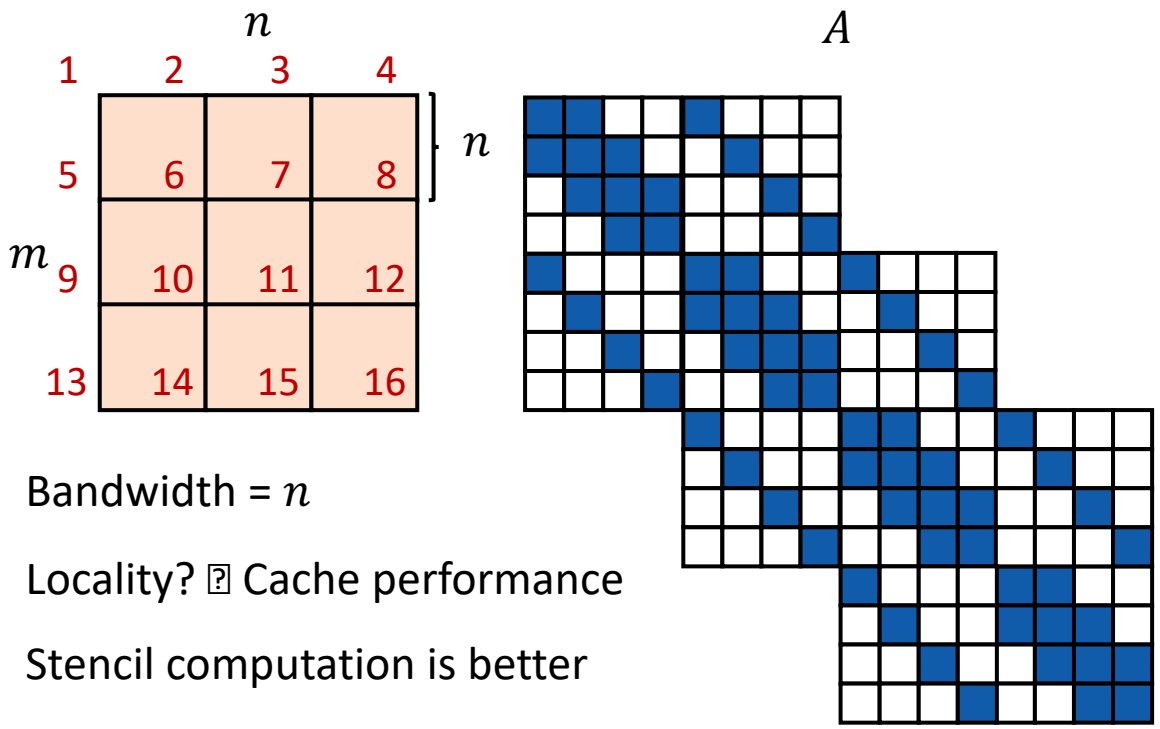
Algebraic Systems of Discretizations

- Different discretization methods might lead to systems with different properties
  - Preserve positive-definiteness and symmetry at discrete level
  - Preserve maximum principle at discrete level
  - Sparsity pattern will affect numerical performance



Reordering Sparse Matrices

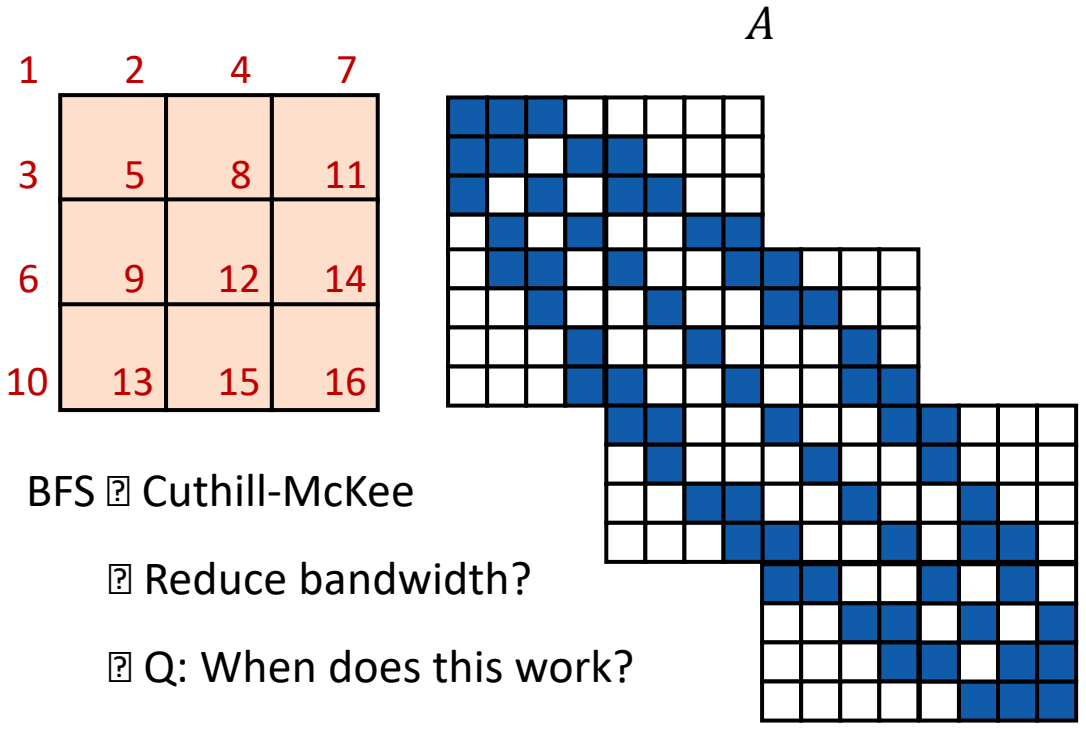
- Sparse matrices are a lot more difficult to deal with and to analyze
- Use different data structures and optimized lib (Lecture by Weifeng Liu)
- Use different ordering to improve efficiency (depends on what solvers you will use)



Bandwidth = n

Locality? ⇒ Cache performance

Stencil computation is better



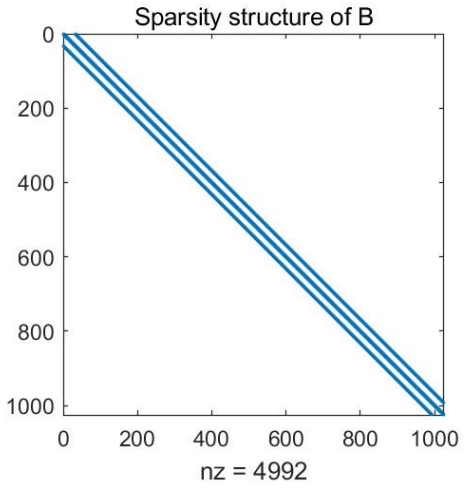
BFS ⇒ Cuthill-McKee

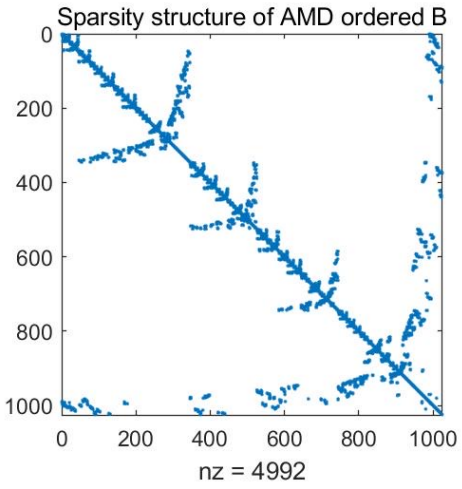
⇒ Reduce bandwidth?

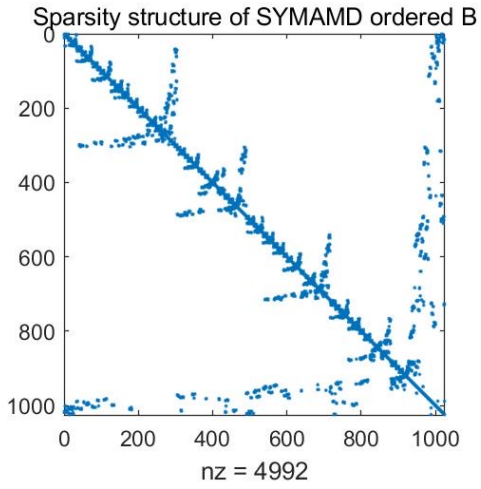
⇒ Q: When does this work?

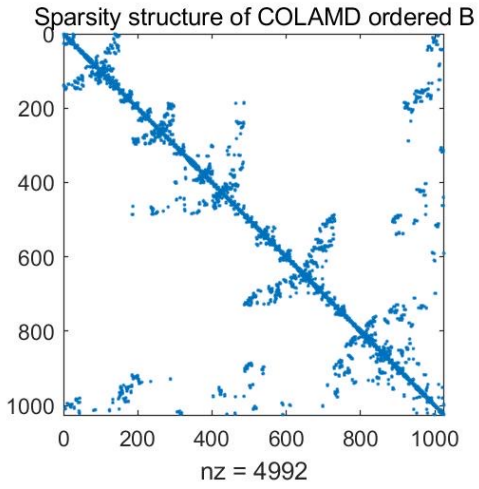
Various Reordering Schemes

Sparsity Patterns

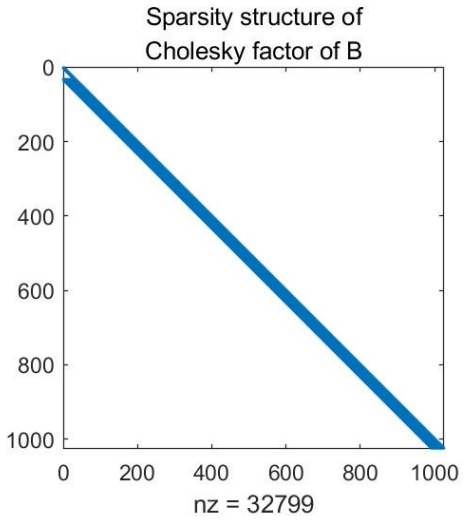


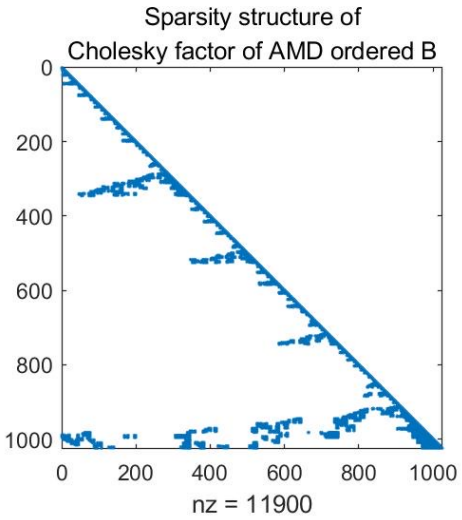


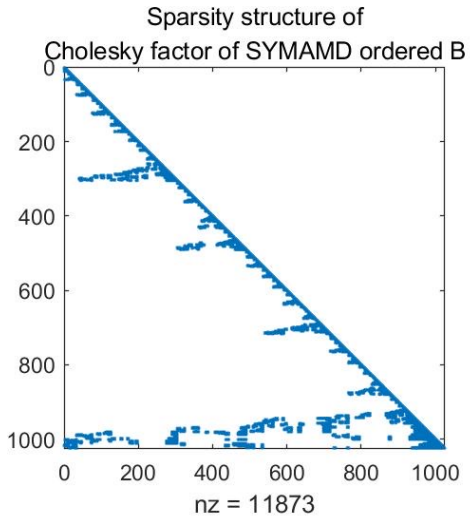


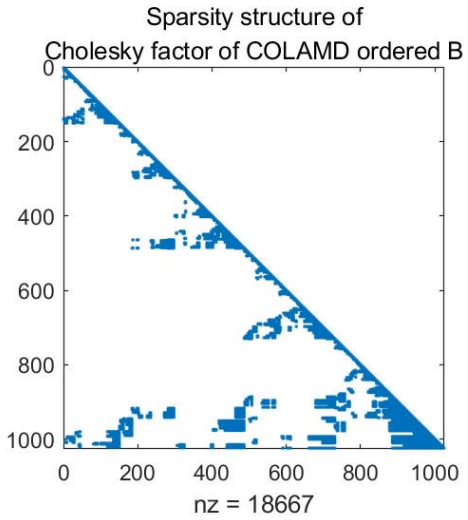


Factorization Patterns



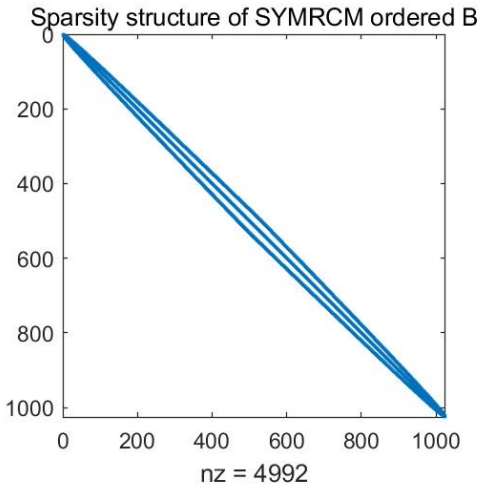


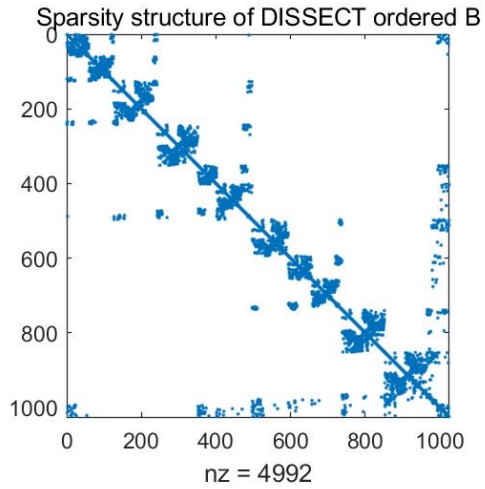


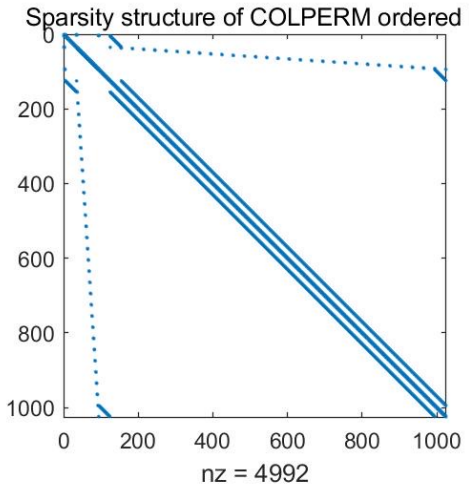


Various Reordering Schemes

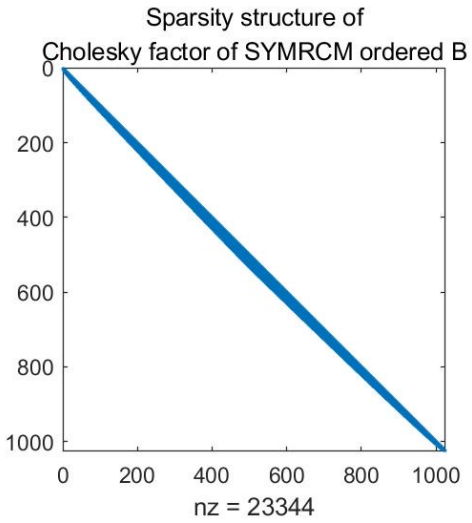
Sparsity Patterns

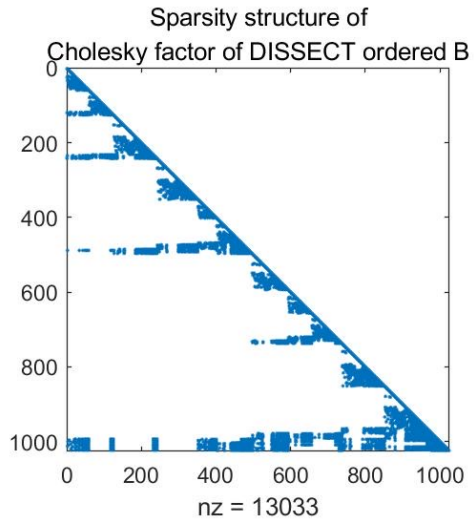


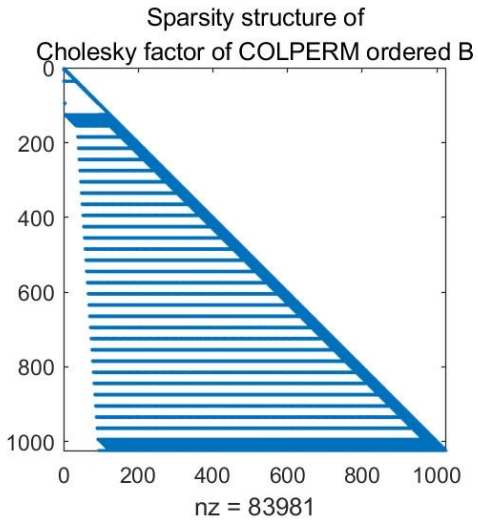




Factorization Patterns







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

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

Computational Complexity

Linearization ⍰ time marching ⍰ spatial discretization ⍰ linear solver ⍰ parallelization

Q: What prevents us from achieving better performance?

Number of nonlinear iterations

Number of time steps

Solver complexity

Complexity $\sim N_{model} \times N_{nl} \times N_{time} \times C_d (N_{space}^d)^k \div P$

How can we achieve this?

Q: Complexity is not wall-time! What else should we consider for real performance?

Number of simulation runs

Number of spatial DoF

Number of processors

Adaptivity + Fast Nonlinear & Linear Solvers + Parallelization

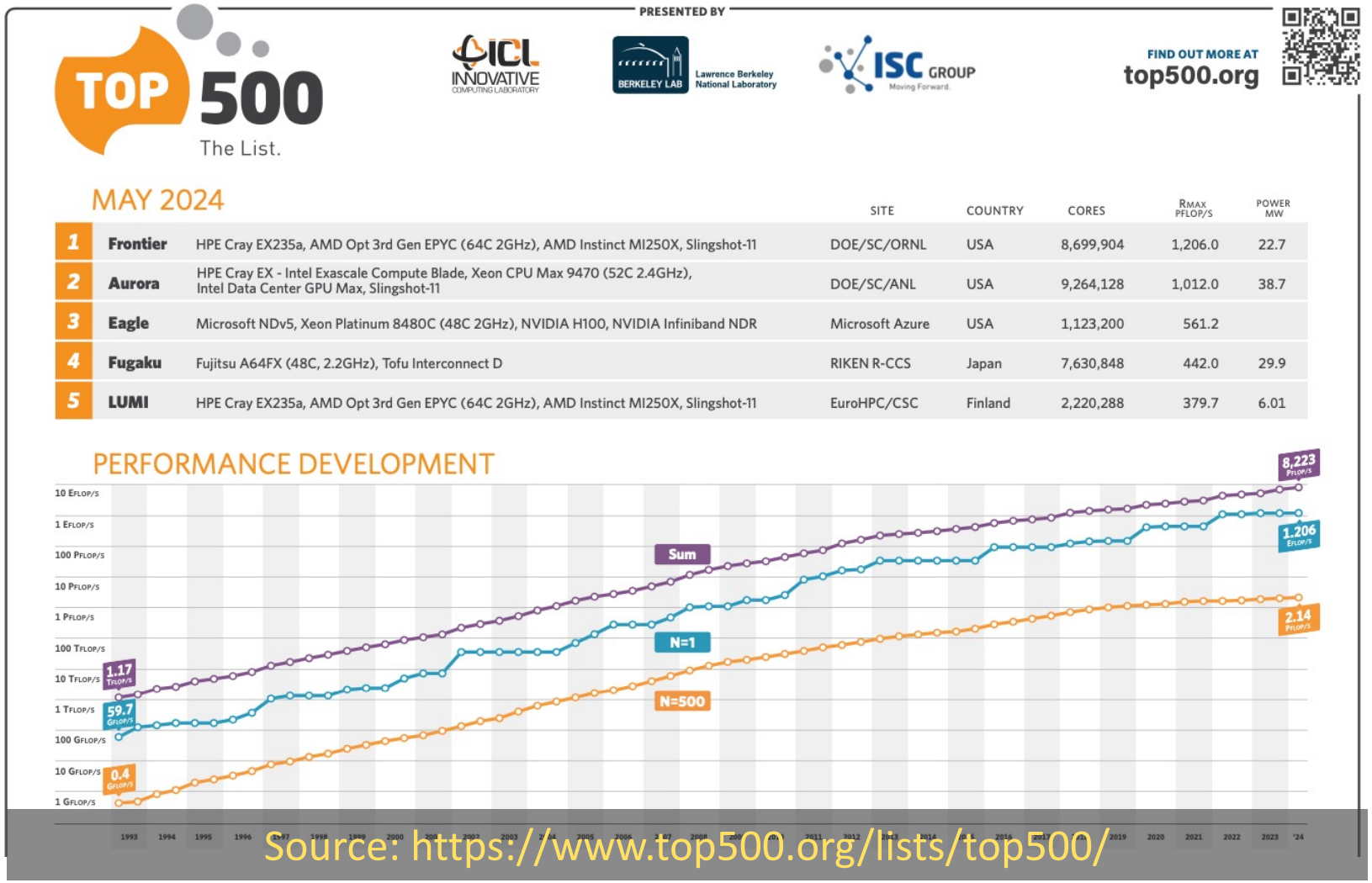
Changes in Large-Scale Computing

Source: 中国工程物理研究院徐小文2019年报告

- 2009-2014：围绕E级计算，DOE组织数十次战略研讨，影响了后续100P/1000P系统的布局以及相关研究计划的部署
- 2015：奥巴马签署国家战略计算规划 (NSCI) 总统令，要求保持美国在超级计算领域的核心竞争力，开始对中国的超算中心实施禁运
- 2016：DOE和NNSA启动百亿亿次计算攻关计划（ECP），全面推进百亿亿次“应用-软件-硬件”协同研发，确保2023年左右实现国家战略安全领域的百亿亿次计算能力
- 2019：DOE签订了三台E级机的采购合同（$18亿硬件研发费用 + $18亿软件开发费用）

Keys: infrastructure, culture, portable, reusable, composable, interoperable, …

HPC Top500 List 2024.06



Source: https://www.top500.org/lists/top500/

- Fancy! Just fancy?
- A lot of cores!!
- Cost a lot of money!!!
- Can this trend continue?
- Do we need HPC?
- Can we use HPC well?
- How to use HPC well?
- HPL (dense/direct)
- HPCG (sparse/iterative)
- HPL-AI (low precision)
- Green 500 (energy)

HPCG and Green500 List 2024.06

Rank	TOP500 Rank	System	Cores	Rmax (PFlop/s)	HPCG (TFlop/s)
1	4	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	16004.50
2	1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,206.00	14054.00
3	2	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	5612.60
4	5	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,752,704	379.70	4586.95
5	6	**Alps** - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE Swiss National Supercomputing Centre (CSCS) Switzerland	1,305,600	270.00	3671.32

https://www.top500.org/lists/hpcg/

Rank	TOP500 Rank	System	Cores	Rmax (PFlop/s)	Power (kW)	Energy Efficiency (GFlops/watts)
1	189	**JEDI** - BullSequana XH3000, Grace Hopper Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, ParTec/EVIDEN EuroHPC/FZJ Germany	19,584	4.50	67	72.733
2	128	**Isambard-AI phase 1** - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE University of Bristol United Kingdom	34,272	7.42	117	68.835
3	55	**Helios GPU** - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE Cyfronet Poland	89,760	19.14	317	66.948
4	328	**Henri** - ThinkSystem SR670 V2, Intel Xeon Platinum 8362 32C 2.8GHz, NVIDIA H100 80GB PCIe, Infiniband HDR, Lenovo Flatiron Institute United States	8,288	2.88	44	65.396
5	71	**preAlps** - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE Swiss National Supercomputing Centre (CSCS) Switzerland	81,600	15.47	240	64.381

https://www.top500.org/lists/green500/

How to Measure Parallel Efficiency

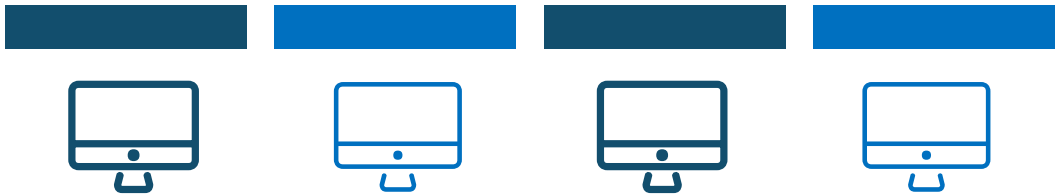
- **强可扩展性**

$$\frac{W_1}{kp_1} \qquad S = \frac{T_1}{T_k} \approx k?$$

若需要解决一个任务，把计算资源增大到原来的 k 倍，能加速 k 倍吗?

定义**并行效率** $E = S/k$，我们希望是100%!

- **弱可扩展性**



$$\frac{W_k}{kp_1} \qquad T_k \approx T_1?$$

若需要解决 k 倍大的任务，计算资源同比例增加，能保持相同耗时吗?

为了**效率**保持不变， 需要多大的问题规模?

Three Mountains on Parallel Simulation

Amdahl's Law 1967

如果串行部分占总时间的10%，那并行加速比不可能超过10倍

对于大规模系统来说，很多应用程序的强可扩展性是难以实现的！

Gustafson-Barsis's Law 1988

对于很多应用来说，一般需要计算越来越大规模的问题！需要是弱可扩展性，而不是强可扩展性

当问题规模与计算资源同比例增大k倍时，加速比最高可达 $0.9k + 0.1$

Gabriel Wittum: HPC Paradox

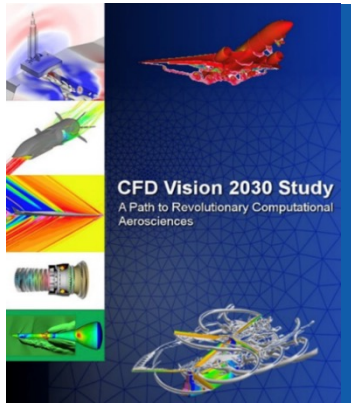
当购买了大10倍的硬件系统，希望能更快地求解大10倍的问题；但现实很残酷，必须有最优算法才可以！

最优算法对于充分发挥HPC效率及实现弱可扩展性至关重要！

硬件投资 >>> 并行效率 >>> 最优算法 >>> 软件投资

Solver is Crucial to Scalability

NASA CFD Vision 2030 Study (2014)



- 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

- 解法器是大规模计算的瓶颈，形成高效、通用、可扩展的解法器是一个公开难题，在一些应用问题的仿真中占用了80%以上的计算时间
- 对超算硬件性能进行排名的Benchmark（如HPL、HPCG等）也常采用线性解法器作为核心指标

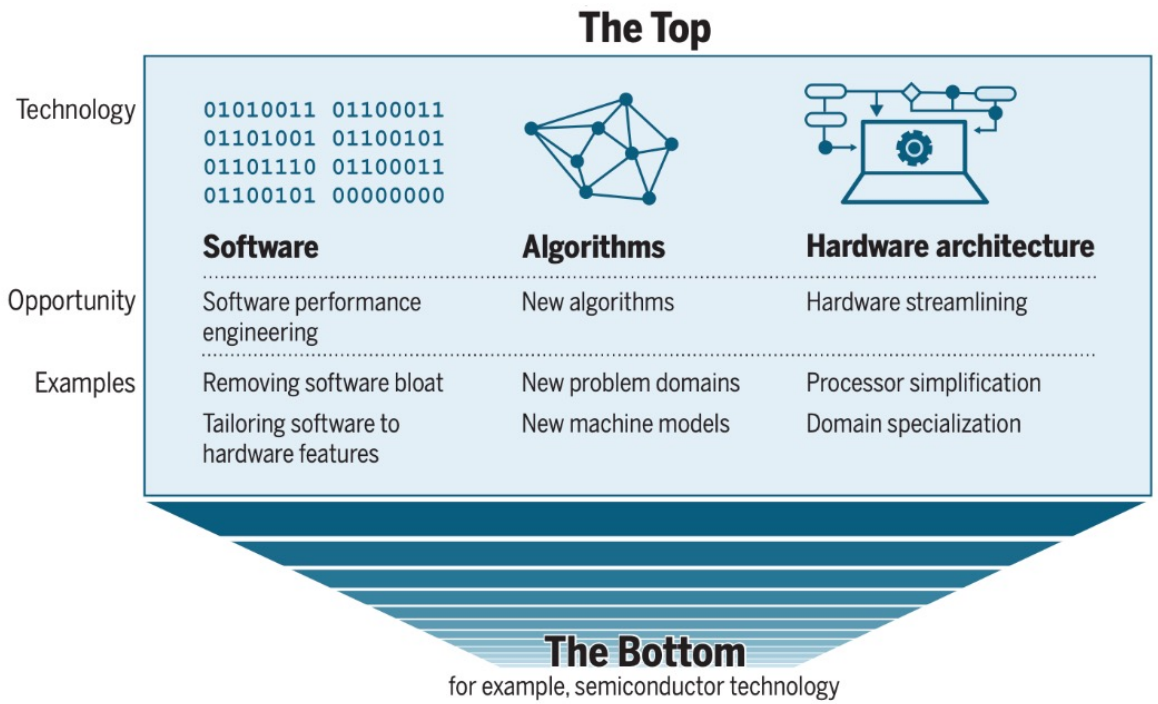
精细模拟 → 并行计算 → 隐式方法 → 线性解法 → 迭代方法 → 预条件子 → 稳健解法 → 可靠解法 → 容错方法

求解大规模问题的典型思路

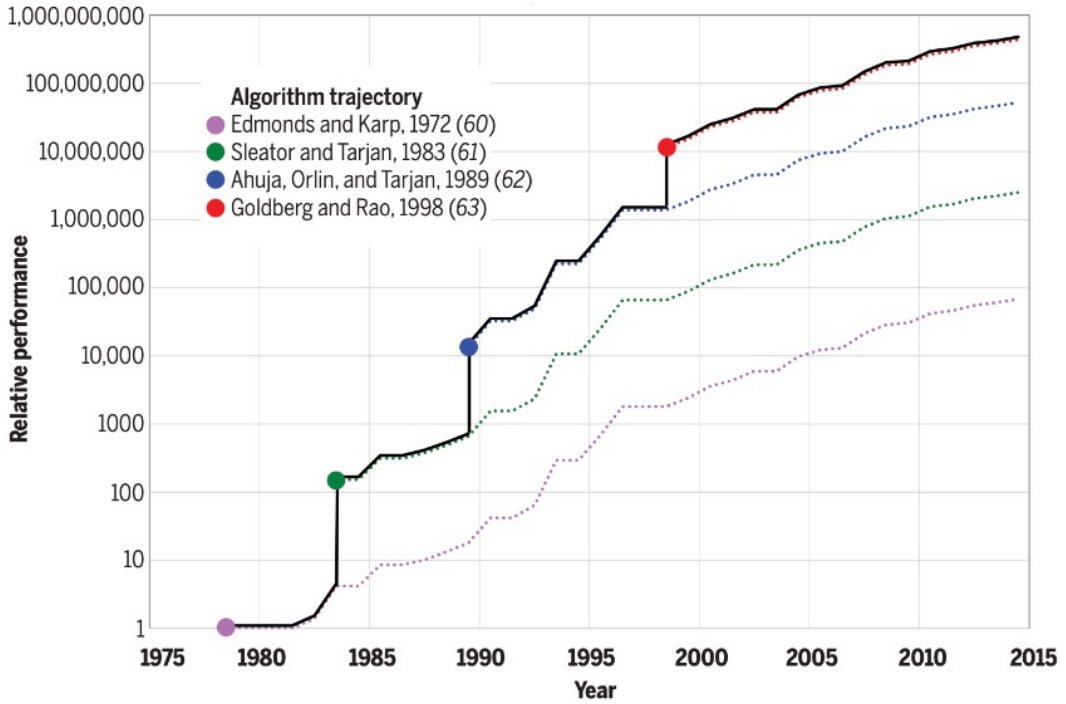
目前急需解决的核心问题

Room for Performance Improvement

- ~~*"There's plenty of room at the bottom"*~~, R. Feynmann, Nobel prize-winner, 1959
- *"There's plenty of room at the top"*, Leiserson, et al., Science 368, 2020



Performance gains after Moore's law ends. In the post-Moore era, improvements in computing power will increasingly come from technologies at the "Top" of the computing stack, not from those at the "Bottom", reversing the historical trend.



Maximum-flow algorithms

Source: arXiv-2203.00671v2, IEEE FOCS 2022

Implementation of Algorithms

Table 1. Speedups from performance engineering a program that multiplies two 4096-by-4096 matrices. Each version represents a successive refinement of the original Python code. "Running time" is the running time of the version. "GFLOPS" is the billions of 64-bit floating-point operations per second that the version executes. "Absolute speedup" is time relative to Python, and "relative speedup," which we show with an additional digit of precision, is time relative to the preceding line. "Fraction of peak" is GFLOPS relative to the computer's peak 835 GFLOPS. See Methods for more details.

Version	Implementation	Running time (s)	GFLOPS	Absolute speedup	Relative speedup	Fraction of peak (%)
1	Python	25,552.48	0.005	1	—	0.00
2	Java	2,372.68	0.058	11	10.8	0.01
3	C	542.67	0.253	47	4.4	0.03
4	Parallel loops	69.80	1.969	366	7.8	0.24
5	Parallel divide and conquer	3.80	36.180	6,727	18.4	4.33
6	plus vectorization	1.10	124.914	23,224	3.5	14.96
7	plus AVX intrinsics	0.41	337.812	62,806	2.7	40.45

366x

174x

MKL

- Compiler optimization; loop ordering; parallel loops
- Tiling; cache-oblivious divide-and-conquer
- Vectorization; AVX intrinsic
- Leiserson & Shun, MIT Open Course 6.172

Source: There's plenty of room at the Top: What will drive computer performance after Moore's law?

Science **368** (6495), June, 2020

http://science.sciencemag.org/content/368/6495/eaam9744

Section 3. Solving Large-scale Linear Systems

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

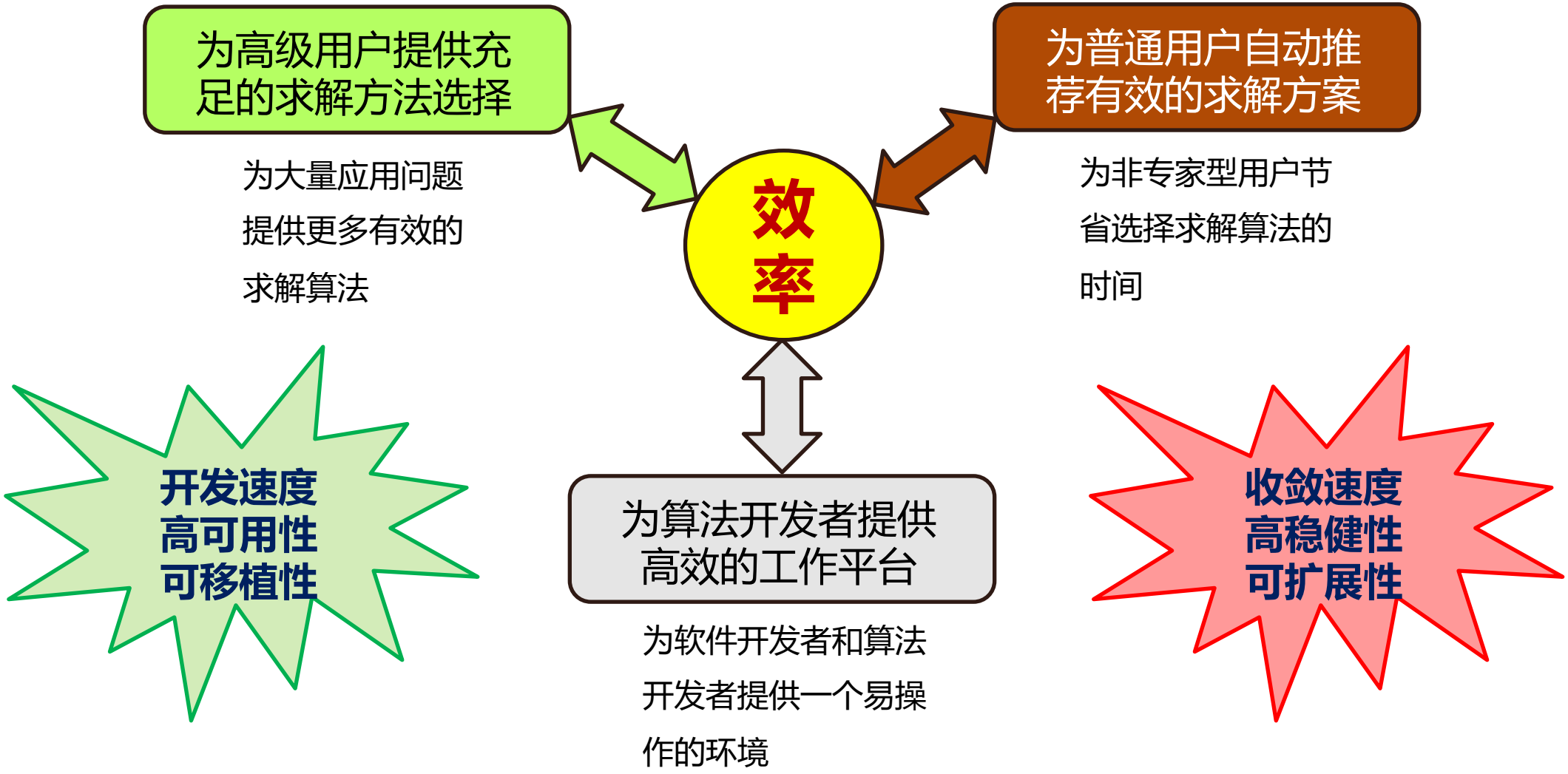
显式方法

一致网格

为什么不使用这样的方法呢？

有时候，身不由己……

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 being the number of grid points in each direction
- Limited applicability: only works for some special partial differential equations and structured grids

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.

Solving Sparse Linear Systems

- Solve a large-scale sparse linear equation: $Ax = b$
- Given an initial guess x. Then define the residual as $r := b - Ax$
- Solve the error equation and correct the initial guess $Ae = r \implies x' = x + e$
- Apply the classical **Richardson** iteration:

$$x' = x + \alpha(b - Ax) \qquad \alpha := \frac{2}{\lambda_1(A) + \lambda_n(A)}$$

- Convergence of error:

$$x_* - x^{(k)} = (I - \alpha A)\left(x_* - x^{(k-1)}\right) \quad \Rightarrow \quad \|x_* - x^{(k)}\| \le \left(\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1}\right)\|x_* - x^{(k-1)}\|$$

- Suppose the initial guess is $x^{(0)} = 0$. Then we have

$$x^{(k)} = \left(I - (I - \alpha A)^k\right)A^{-1}b =: p_{k-1}(A)b \approx A^{-1}b$$

Iterative Methods

- A very long history: Newton, Euler, Guass, ...
- Linear: Gauss-Seidel, SOR, Krylov subspace methods, domain decomposition, multigrid, ...
- Nonlinear: Steepest descent method, Newton's method, power iteration, inverse iteration, ...
- 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 step-size

Steepest Descent Method

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

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

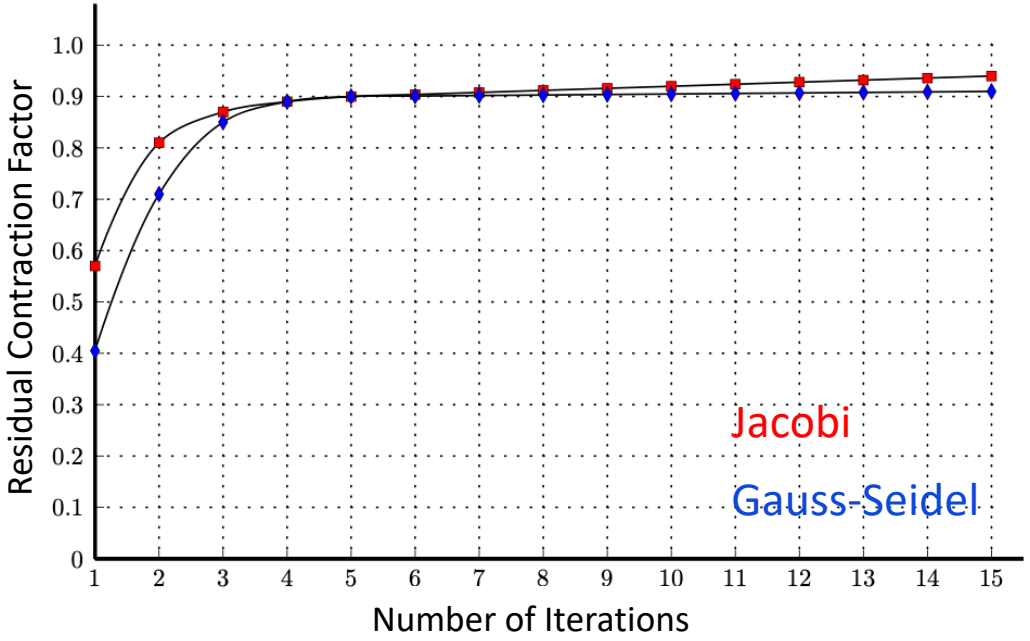
General Iterative Linear Solvers

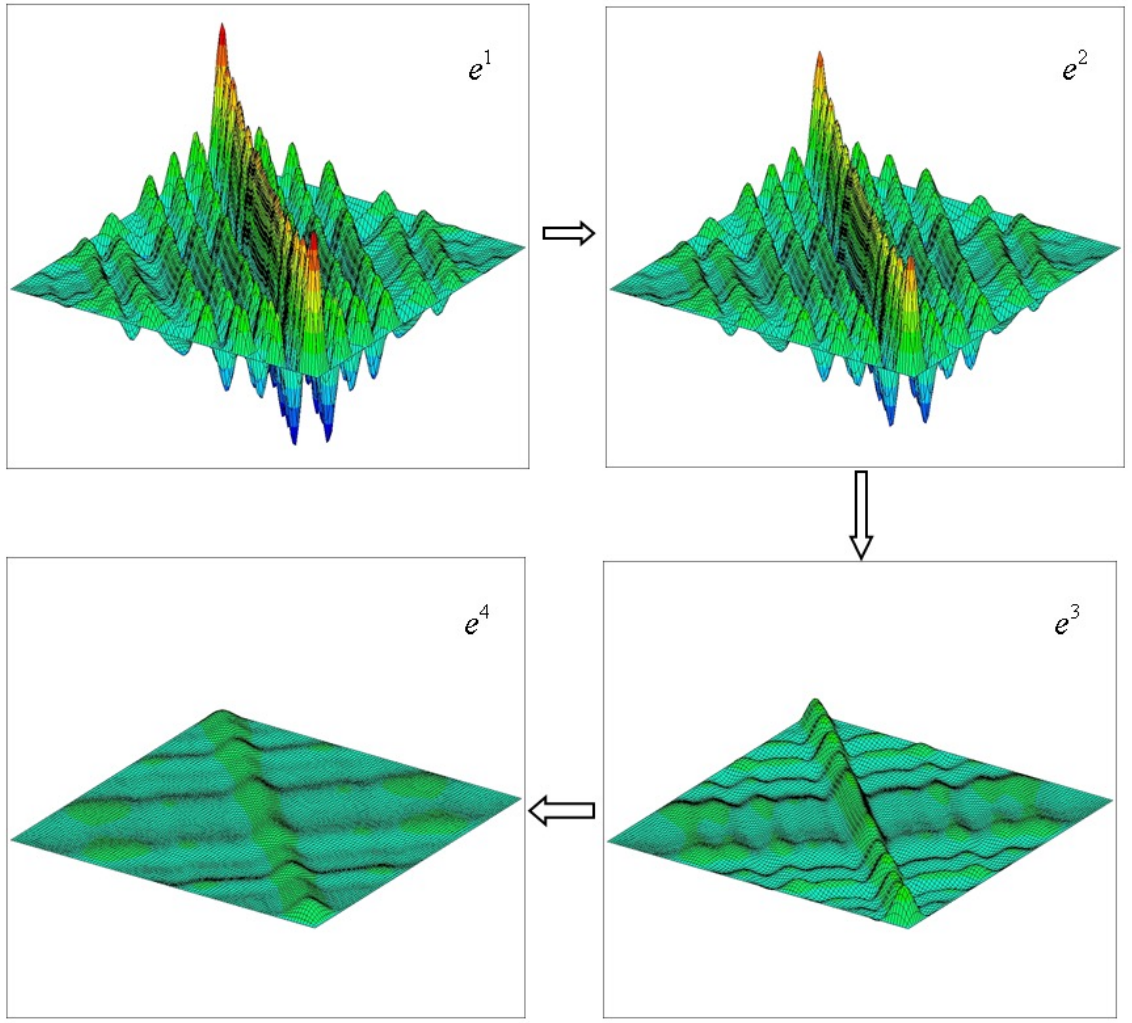
Algorithm 1: Iterative solver

```
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
```

- Q: How to implement **Line 4** in practice? Still the same problem!
- Simple approximations: Jacobi, weighted Jacobi, block Jacobi, Gauss-Seidel, SOR, …
- Use the same method for Line 4 ⇒ Linear stationary methods
- Line 4 alone ⇒ Preconditioner
- Iterative refinement (IR) methods

Simple Iterative Methods





- Simple iterative methods usually slow down after a few iterations (relaxation stage)
- Local relaxation methods have smoothing properties but cannot deal with global error

Fast Poisson Solvers on CPUs

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。

Fast Poisson Solvers on GPUs

Grid	#DOFs	FMG(1,1)	L^2 Error	FMG(3,3)	L^2 Error	FMG(5,5)	L^2 Error
1024^2	1M	1.955e-2 s	2.618e-6	2.316e-2 s	1.790e-7	2.711e-2 s	2.541e-7
2048^2	4M	2.288e-2 s	6.766e-7	2.762e-2 s	4.479e-8	3.410e-2 s	6.356e-8
4096^2	16M	3.311e-2 s	1.735e-7	3.987e-2 s	1.120e-8	4.956e-2 s	1.589e-8
8192^2	64M	5.761e-2 s	4.421e-8	7.958e-2 s	2.801e-9	1.027e-1 s	3.974e-9
16384^2	256M	**1.346e-1 s**	1.122e-8	2.073e-1 s	7.002e-10	2.831e-1 s	9.905e-10

Grid	#DOFs	FMG(1,1)	L^2 Error	FMG(3,3)	L^2 Error	FMG(5,5)	L^2 Error
64^3	0.26M	1.139e-2 s	2.685e-3	1.310e-2 s	1.608e-4	1.749e-2 s	5.462e-5
128^3	2M	1.325e-2 s	1.032e-3	4.531e-2 s	4.394e-5	1.871e-2 s	1.488e-5
256^3	16M	1.751e-2 s	3.803e-4	2.605e-2 s	1.145e-5	3.837e-2 s	3.844e-6
512^3	128M	5.531e-2 s	1.364e-4	9.658e-2 s	2.917e-6	1.294e-1 s	9.737e-7
1024^3	1024M	3.792e-1 s	4.805e-5	**7.153e-1 s**	7.358e-7	1.003e+0 s	2.447e-7

Poisson方程（均匀网格差分格式）的线性解法器对比，CPU: Intel Xeon8358，GPU: NVIDIA A100（张林杰，2024.07）

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

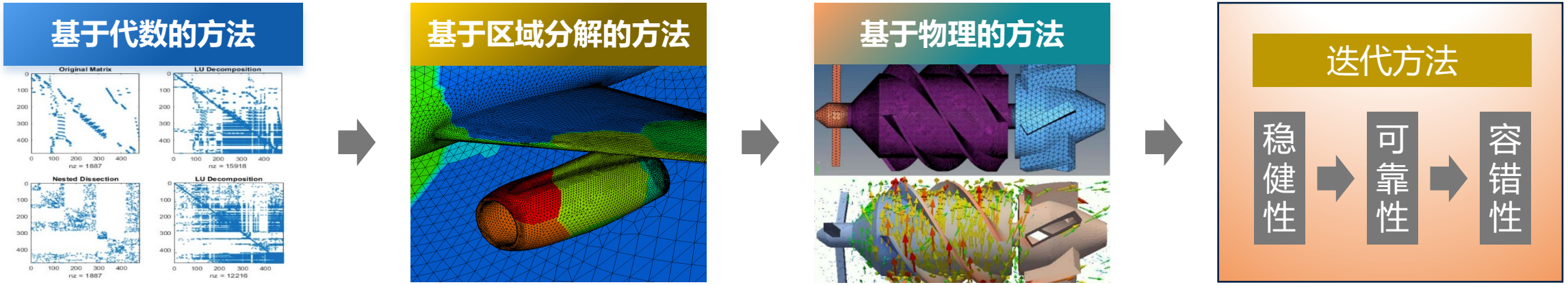
Direct solvers > Iterative solvers

- 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 difference in practice

Method of Choice

The Method of Choice

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



Chensong Zhang, AMSS

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

Release version 2024.09.06