2024年秋季·多层迭代法
课程内容总结

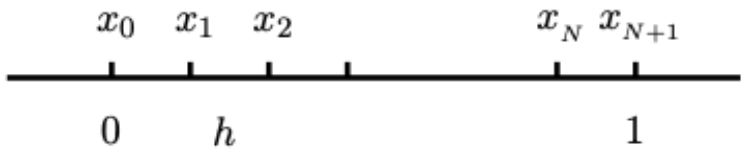
日期：2024.11.28

教师：张晨松，中国科学院数学与系统科学研究院

模型问题：求解Poisson方程

模型方程及其有限差分离散：

$$
\begin{aligned}
-u''(x) &= f, \qquad x \in (0,1) \\
u(x) &= 0, \qquad x = 0,1
\end{aligned}
$$



$$
\begin{aligned}
u''(x_j) &= \frac{1}{h^2}\left[u(x_{j-1}) - 2u(x_j) + u(x_{j+1})\right] + O(h^2) \\
&\approx \frac{1}{h^2}\left[u_{j-1} - 2u_j + u_{j+1}\right],
\end{aligned}
$$

$$
\vec{f} := \left(f_j\right)_{j=1}^{N} = \left(f(x_j)\right)_{j=1}^{N}
$$

问题特点：

- 常用模型问题：方程足够简单，但包含了很多重要应用的难点特征
- 应用范围广泛：Heat conduction, fluid flow, porous-media flow, ……
- 问题条件数大：条件数随网格尺寸平分反比增长，ill-conditioned
- 实际应用问题：**各向异性**、**跳跃系数**、**非均质性**、……

可用方法：

- 追赶法，LU
- FFT
- CG
- **Multigrid**

简单迭代法的主要思想

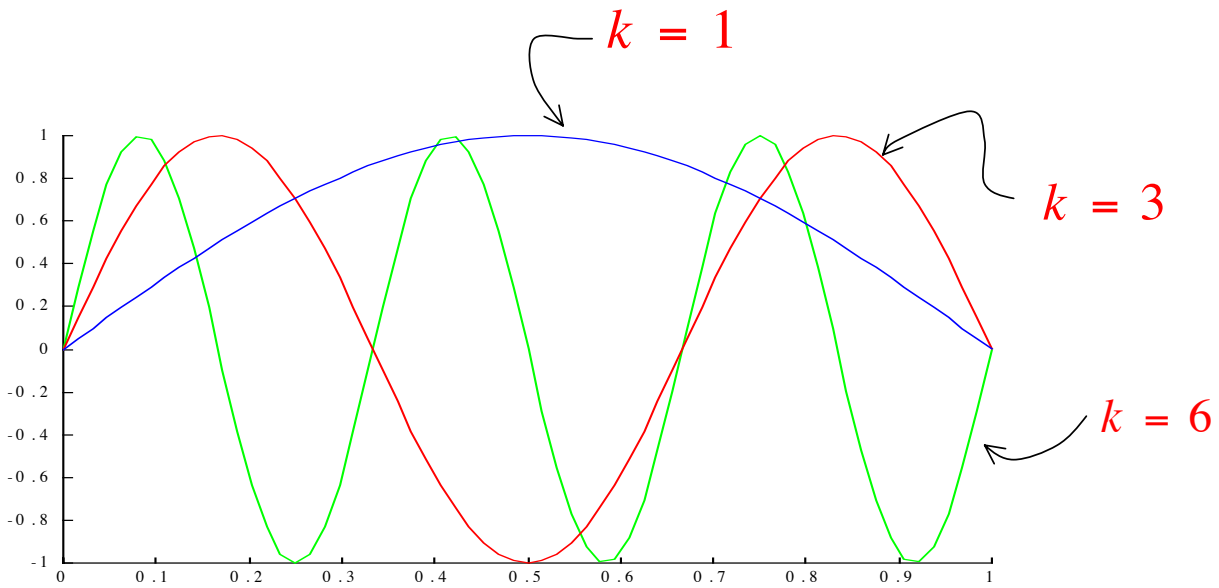
Algorithm 1: Iterative solver

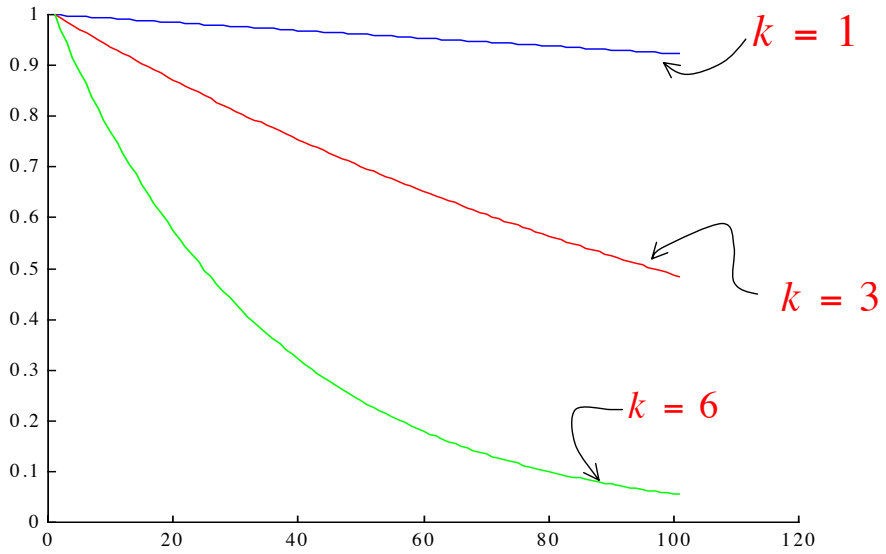
```
%% Given a nonsingular matrix A ∈ R^{n×n}, b ∈ R^n, and an initial guess x_0 ∈ R^n;
for i = 0 : MaxIter or converged
    Compute r_i ← b − A x_i;
    Solve the error equation A e_i = r_i approximately;
    Update x_{i+1} ← x_i + e_i;
end
```

简单迭代：

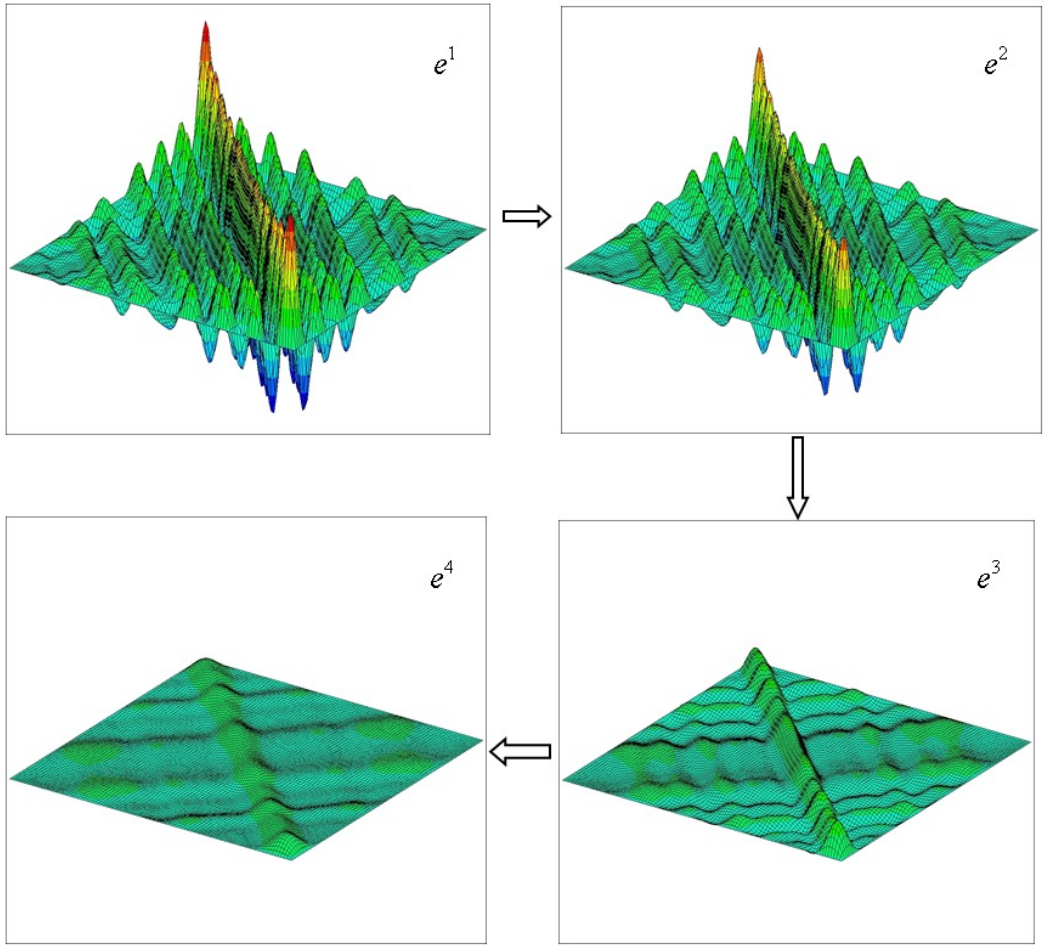
$$x \leftarrow x + S(b - Ax)$$

- wJacobi或GS方法等
- 程序非常简单
- 收敛速度极慢





两层网格法的主要思想



仅使用磨光算法的效果

两层网格方法（TG）：

- Step 1. 前磨光（简单迭代法）

$$x \leftarrow x + S(b - Ax)$$

- Step 2. 粗空间校正（CGC）

$$x \leftarrow x + PA_c^{-1}R(b - Ax)$$

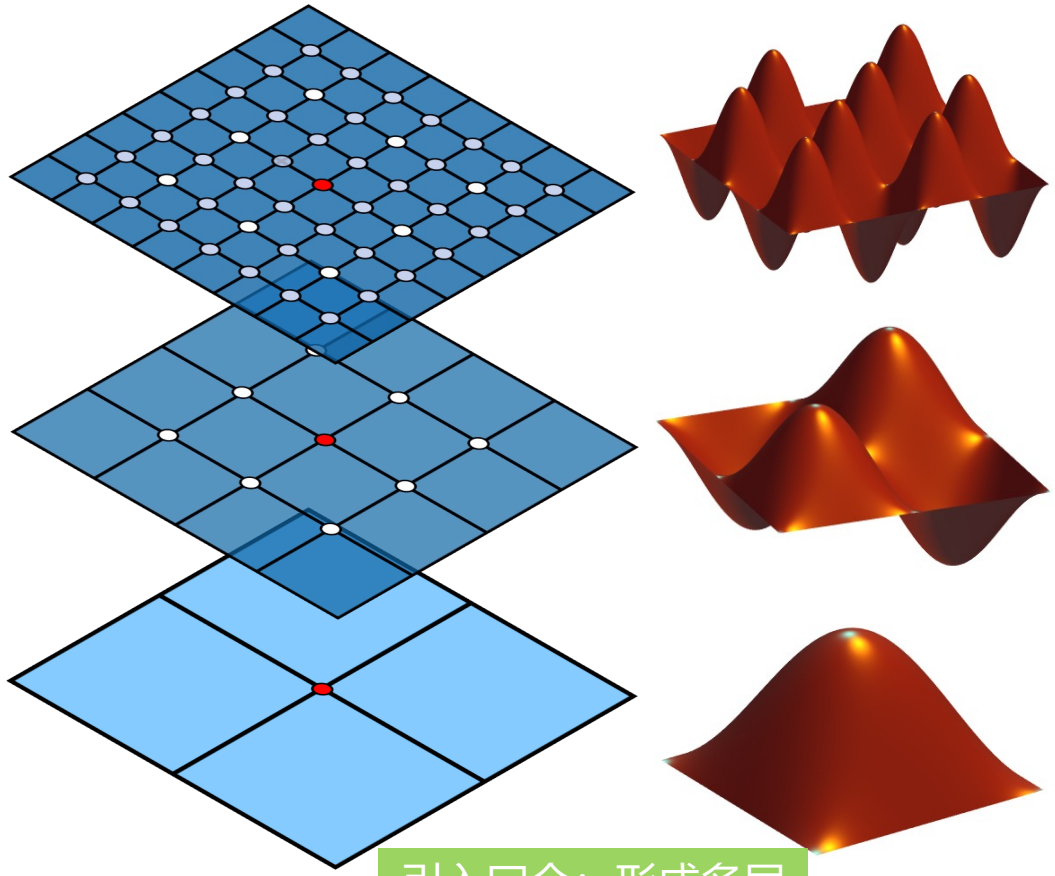
两层网格方法误差传播算子：

$$\begin{aligned} E_{\mathrm{TG}} &= (I - \Pi_c)(I - SA) \\ &= (I - PA_c^{-1}RA)(I - SA) \end{aligned}$$

利用磨光消除高频误差，利用粗网格校正CGC消除低频误差

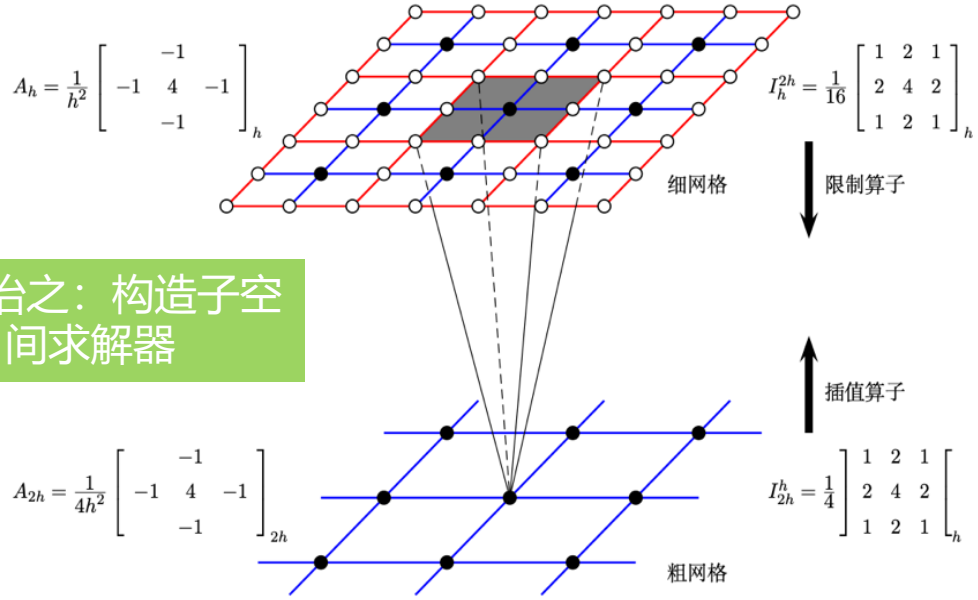
多层迭代法的主要思路

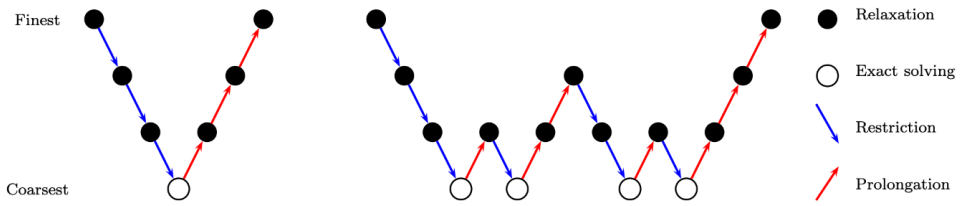
问题：如何处理低频误差分量？如何有效消除全局误差的影响？如何高效地进行实现？



引入冗余：形成多层子空间分解

分而治之：构造子空间求解器





分层迭代：有效的嵌套迭代格式

多层网格V-循环伪代码

```
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 ← S 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 + S^T r_ℓ; %% post-smoothing (optional)
12 end
13 return x_ℓ
```

简单示例：1D几何多层网格法

1D Poisson方程的GMG：

- FDM五点差分模板
- 需要层次化结构网格
- 可采用无矩阵实现方法，效率高

$$A = \frac{1}{h^2}\,\mathrm{tridiag}(-1, 2, -1),\ f_i = f(x_i).$$

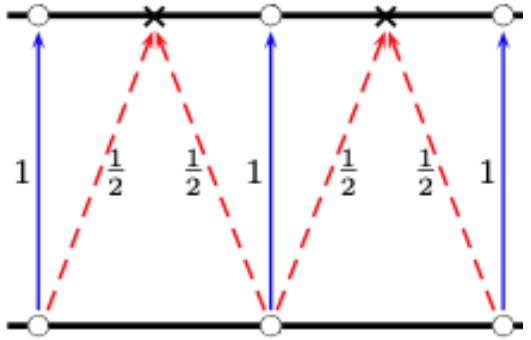
$\vdots$

$l = 2$ —— $h_2 = (\frac{1}{2})^3$

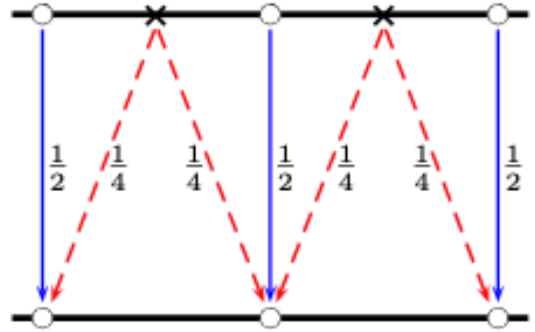
$l = 1$ —— $h_1 = (\frac{1}{2})^2$

$l = 0$ —— $h_0 = (\frac{1}{2})^1$

$P_{l-1,l}$



$R_{l,l-1}$



$$P_{l-1,l} := \frac{1}{2}\begin{pmatrix} \ddots & & & & \\ & 1 & & & \\ & 2 & & & \\ & 1 & 1 & & \\ & & 2 & & \\ & & 1 & & \\ & & & \ddots & \end{pmatrix} \qquad R_{l,l-1} := \frac{1}{4}\begin{pmatrix} \ddots & & & & & & \\ & 1 & 2 & 1 & & & \\ & & & 1 & 2 & 1 & \\ & & & & & & \ddots \end{pmatrix}$$

粗网格问题：$A_c = \left(\frac{h}{H}P^T\right)AP =: RAP.$

或可直接由粗网格进行离散获得

多层网格法的简单示例

I need a geometric multigrid solver for the Poisson's equation in 2D. Can you write a MATLAB code for me?

```
function gmg_poisson_2d()
    % Grid parameters
    n = 64; % Number of grid points (must be power of 2)
    L = 1;  % Length of the domain
    h = L / (n - 1); % Grid spacing

    % Generate grid
    [x, y] = meshgrid(0:h:L, 0:h:L);

    % Define the right-hand side
    f = sin(pi * x) .* sin(pi * y);

    % Initial guess
    u = zeros(size(f));

    % Solve using GMG
    num_cycles = 10; % Number of V-cycles
    u = v_cycle(u, f, h, num_cycles);

    % Plot solution
    figure;
    surf(x, y, u, 'EdgeColor', 'none');
    title('Solution to Poisson Equation');
    xlabel('x');
    ylabel('y');
    zlabel('u');
    colorbar;
end
```

```
function u = mg_step(u, f, h, level)
    % Multigrid recursive V-cycle step
    if level == 0
        % Coarsest level: solve directly
        u = solve_coarsest(f, h);
    else
        % Pre-smoothing
        u = gauss_seidel(u, f, h, 3);

        % Compute residual
        residual = f - laplacian(u, h);

        % Restrict residual to coarse grid
        residual_coarse = restrict(residual);

        % Zero initial guess for error on coarse grid
        error_coarse = zeros(size(residual_coarse));

        % Recursively solve for error
        h_coarse = 2 * h;
        error_coarse = mg_step(error_coarse, residual_coarse, h_coarse, level - 1);

        % Prolongate error to fine grid and correct
        error_fine = prolong(error_coarse);
        u = u + error_fine;

        % Post-smoothing
        u = gauss_seidel(u, f, h, 3);
    end
end
```

写一个多层网格法的简单示例

```
function u = v_cycle(u, f, h, num_cycles)
    % Perform multigrid V-cycle
    for cycle = 1:num_cycles
        u = mg_step(u, f, h, log2(size(u, 1)) - 1);
    end
end

function u = gauss_seidel(u, f, h, num_sweeps)
    % Gauss-Seidel relaxation
    n = size(u, 1);
    for sweep = 1:num_sweeps
        for i = 2:n-1
            for j = 2:n-1
                u(i, j) = 0.25 * (h^2 * f(i, j) + u(i-1, j) + u(i+1, j) + u(i, j-1) + u(i, j+1));
            end
        end
    end
end

function res = laplacian(u, h)
    % Compute the discrete Laplacian of u
    res = (circshift(u, [-1, 0]) + circshift(u, [1, 0]) + circshift(u, [0, -1]) + circshift(u, [0, 1]) - 4 * u) / h^2;
end

function u_coarse = restrict(u)
    % Restriction operator (full-weighting)
    n_coarse = (size(u, 1) - 1) / 2 + 1;
    u_coarse = zeros(n_coarse);
    for i = 2:n_coarse-1
        for j = 2:n_coarse-1
            u_coarse(i, j) = 0.25 * u(2*i-1, 2*j-1) + ...
                             0.125 * (u(2*i, 2*j-1) + u(2*i-2, 2*j-1) + u(2*i-1, 2*j) + u(2*i-1, 2*j-2)) + ...
                             0.0625 * (u(2*i, 2*j) + u(2*i, 2*j-2) + u(2*i-2, 2*j) + u(2*i-2, 2*j-2));
        end
    end
end
```

```
function u_fine = prolong(u)
    % Prolongation operator (bilinear interpolation)
    n_fine = 2 * (size(u, 1) - 1) + 1;
    u_fine = zeros(n_fine);
    for i = 1:size(u, 1)
        for j = 1:size(u, 1)
            u_fine(2*i-1, 2*j-1) = u(i, j);
        end
    end
    for i = 1:size(u, 1)-1
        for j = 1:size(u, 1)-1
            u_fine(2*i, 2*j-1) = 0.5 * (u(i, j) + u(i+1, j));
            u_fine(2*i-1, 2*j) = 0.5 * (u(i, j) + u(i, j+1));
            u_fine(2*i, 2*j) = 0.25 * (u(i, j) + u(i+1, j) + u(i, j+1) + u(i+1, j+1));
        end
    end
end

function u = solve_coarsest(f, h)
    % Solve on coarsest grid using direct method
    n = size(f, 1);
    A = gallery('poisson', n - 2) / h^2;
    f_vec = reshape(f(2:end-1, 2:end-1), [], 1);
    u_vec = A \ f_vec;
    u = zeros(n);
    u(2:end-1, 2:end-1) = reshape(u_vec, n-2, n-2);
end
```

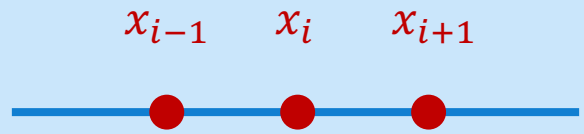
Why do you still write a code like an ancient man? Why do you still write a code anyways?

Can you tell which code is better when you see it? Can you tell AI what kind of code you need?

几何多层网格法的实现

无矩阵GMG实现方式：

- 存储量大幅度降低，减少内存使用量
- 可对相似运算重排/合并，提高cache命中率
- 可对部分模块进行细粒度并行优化



$$A[i,:]\,u = -1 * u_{i-1} + 2 * u_i - 1 * u_{i+1}$$

Fixed stencil computation

提高GMG实现的通用性：

- 基于稀疏矩阵运算进行，通用性强
- 与代数多层网格法（AMG）的结构相同
- 利用稀疏矩阵运算库（如SpMV内核）

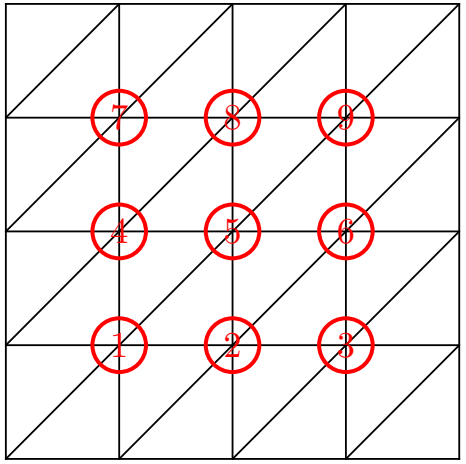
$$u = u + \omega D^{-1}(f - Au)$$

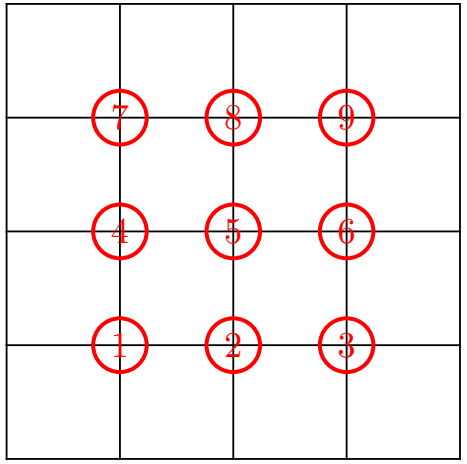
$$u = u + P(RAP)^{-1}R(f - Au)$$

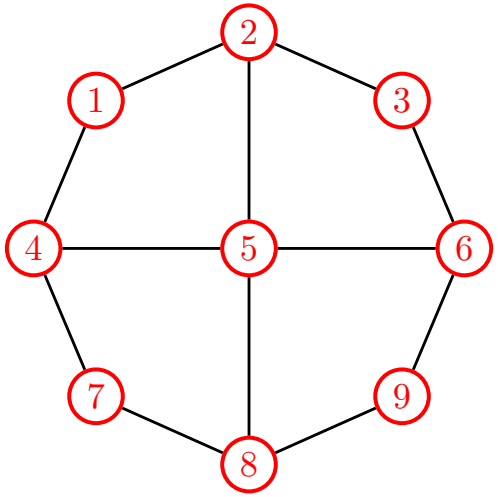
从GMG到AMG

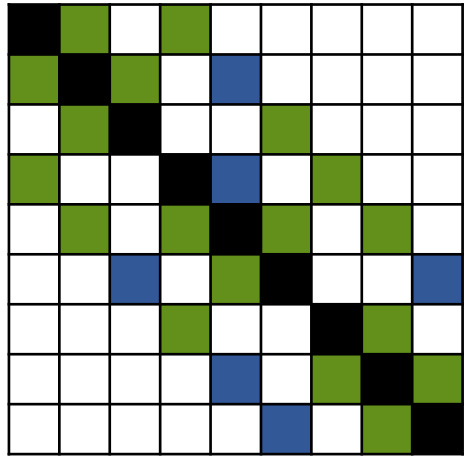
GMG在实际应用中的困难：

- 网格限制多：效率依赖于结构化嵌套加密网格的信息，实际应用中网格往往无法满足要求
- 通用性不高：程序是问题相关的（白盒子），需要case-by-case地设计方法
- 应用难度大：很难应用在有各种限制条件的实际问题中









- 系数矩阵可以对应一个图（带权或者不带权），对称问题 ➔ 无向图，非对称问题 ➔ 有向图
- 可根据临接图反推方程或离散方法的部分信息，构造operator-dependent方法 ➔ 代数多层网格法

经典代数多重网格法C-AMG

Strong n-coupling: $-a_{i,j} \geqslant \theta_{\text{str}} \left| \min_k a_{i,k} \right|$

- $A = (a_{i,j}) \in \mathbb{R}^{N \times N}$ is an M-matrix
- $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ is the corresponding graph of A
- $\mathsf{S}_i := \{ j \in \mathsf{N}_i : \ j \ \text{strongly coupled to} \ i \}$
- $\mathsf{S}_i^T := \{ j \in \mathsf{V} : \ i \in \mathsf{S}_j \}$, set affected by i

```
1  U ← V,  C ← ∅,  F ← ∅;
2  while U ≠ ∅
3        λ_i ← 2|S_i^T ⋂ F| + |S_i^T ⋂ U|,  i ∈ U;
4        k ← arg max{λ_i, i ∈ U};
5        C ← C⋃{k},  U ← U\{k};
6        F ← F⋃S_k^T,  U ← U\S_k^T;
7  end
```

$$A = \begin{pmatrix} A_{ff} & A_{fc} \\ A_{cf} & A_{cc} \end{pmatrix} \quad P := \begin{pmatrix} W \\ I \end{pmatrix}$$

$$a_{ii} e_i + \sum_{j \in \mathsf{N}_i} a_{ij} e_j = 0, \quad i \in \mathsf{F}.$$

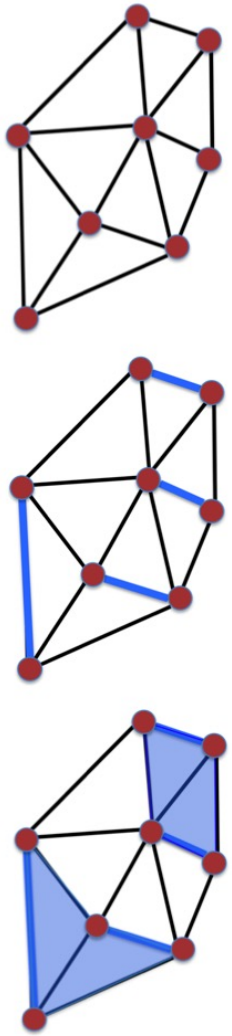
$$a_{ii} e_i + \alpha_i \sum_{j \in \mathsf{N}_i \cap \mathsf{C}} a_{ij} e_j = 0,$$

where $\alpha_i := \dfrac{\sum_{k \in \mathsf{N}_i} a_{ik}}{\sum_{k \in \mathsf{N}_i \cap \mathsf{C}} a_{ik}}$

$$W = \operatorname{diag}(A_{fc} \mathbf{1}_c)^{-1} A_{fc}.$$

Direct Interpolation

聚集代数多重网格法A-AMG



- Tentative prolongation

$$P_{ij} := \begin{cases} 1, & i \in \mathcal{A}_j \\ 0, & i \notin \mathcal{A}_j \end{cases} \qquad P := \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}, \qquad R := P^T = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

- Coarser problem matrix

$$A_c := RAP = P^T AP$$

- 算法的复杂度低，但收敛性受到影响
  - 利用光滑聚集方法SA-AMG
  - 利用更精确的CGC方法：W-cycle, AMLI-cycle, Krylov-cycle

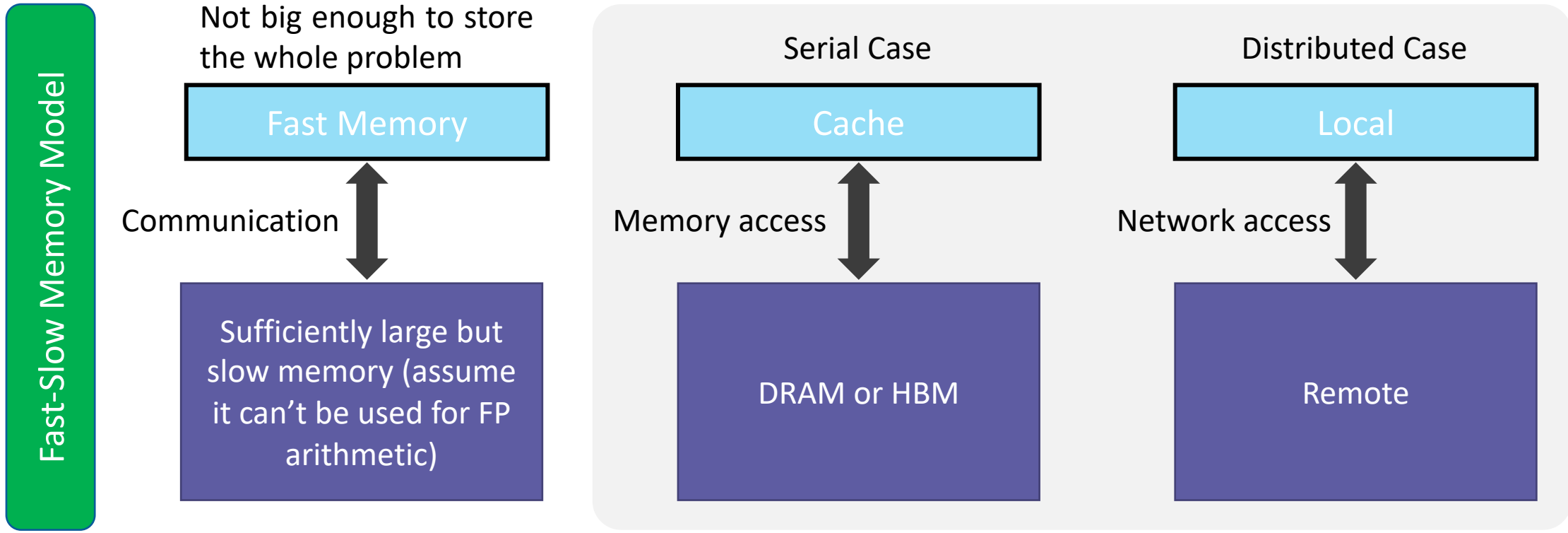
数值模拟的重要内核：快速Poisson解法器

问题规模	网格剖分	64^3			128^3	256^3	512^3	1024^3
	变量个数	0.25M			2M	16M	135M	1024M
稀疏直接法@北京超算	计算核数	8x1	16x1	32x1	16x8	16x64	16x512	16x1024
	求解时间	**5.38s**	**3.86s**	**3.26s**	**59.78s**	**999.46s**	**/**	**/**
GMG@CPU	求解时间	**0.030s**			**0.303s**	**2.815s**	**23.54s**	**/**
GMG@GPU	求解时间	**0.013s**			**0.026s**	**0.045s**	**0.097s**	**0.715s**

三维Poisson方程（均匀网格七点差分格式）的线性解法器对比，稀疏直接法软件Intel MKL Pardiso（北京超级云超算）和几何多重网格法FASP（笔记本电脑、nVidia A100）：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。单节点配两颗32核AMD EPYC7452，256GB内存。

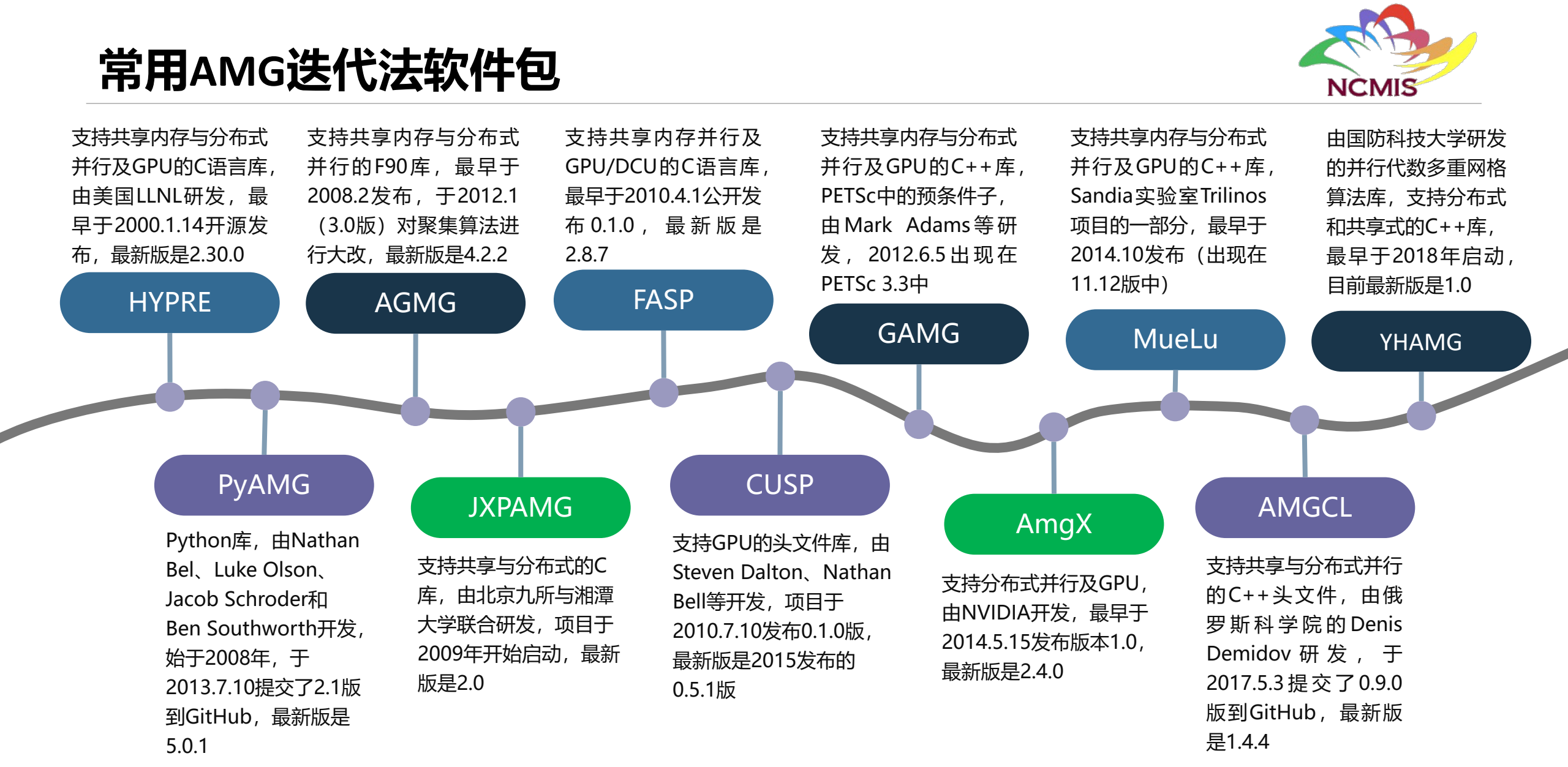
考虑通信开销的性能模型

Communication Time = **Latency** + Num of Bytes Moved ÷ **Bandwidth**



Ref: CS267 lecture notes on J. Demmel's webpage: https://people.eecs.berkeley.edu/~demmel/

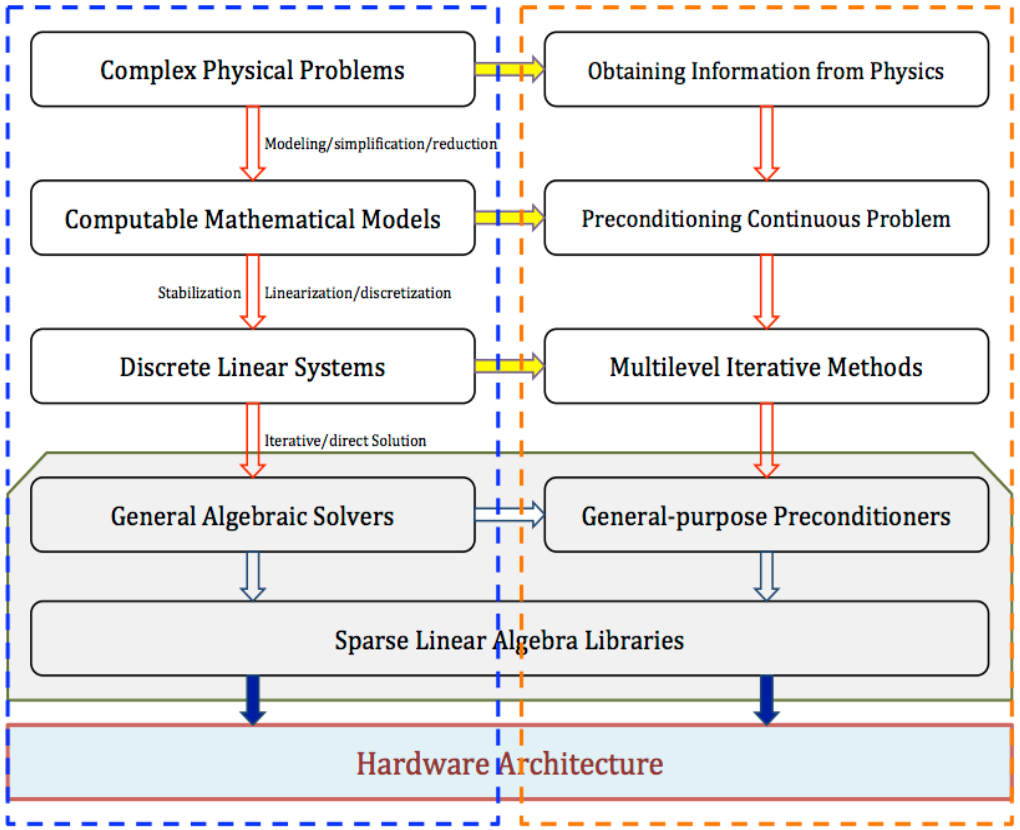
常用AMG迭代法软件包



开源解法器FASP项目

FASP开源项目网址

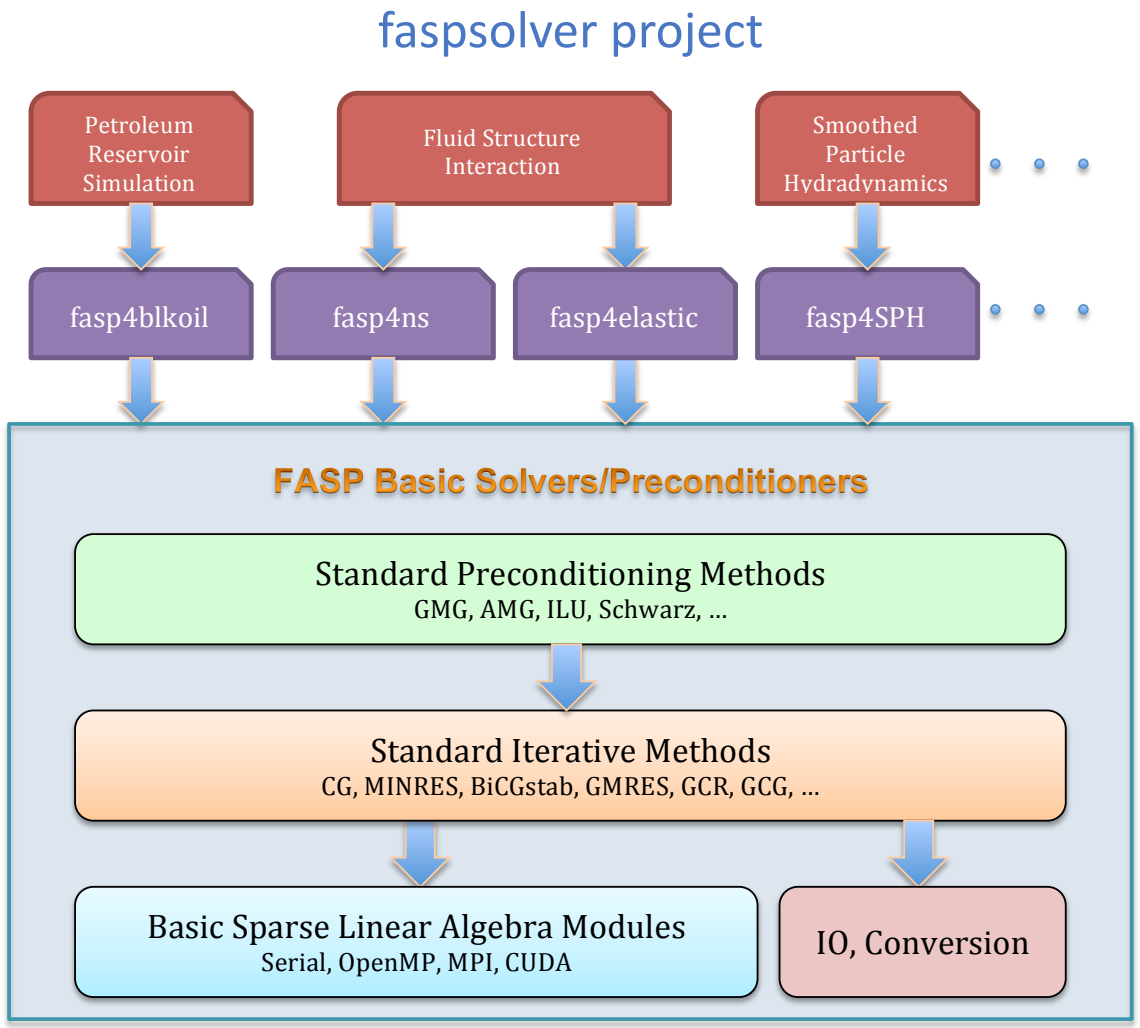
http://www.multigrid.org/fasp
https://github.com/FaspDevTeam/faspsolver
https://gitee.com/faspdevteam/faspxx



FASP及相关开源软件项目：

- faspsolver，开源迭代解法器项目
- fasp4ns，开源流体解法器项目
- fasp4cuda，开源GPU解法器项目
- fasp4dcu，开源国产GPU解法器项目
- fasp4blkoil，开源油藏模拟解法器项目
- faspxx，开源并行迭代解法器项目
- StructMG，拓扑结构化网格求解器项目
- SemiStructMG，半结构化网格求解器项目

faspsolver解法器软件简介



Fast Auxiliary Space Preconditioning (FASP) Solver Library: README

Introduction

The FASP package is designed for developing and testing new efficient solvers and preconditioners for discrete partial differential equations (PDEs) or systems of PDEs. The main components of the package are standard Krylov methods, algebraic multigrid methods, and incomplete factorization methods. Based on these standard techniques, we build efficient solvers, based on the framework of Auxiliary Space Preconditioning, for several complicated applications. Current examples include the fluid dynamics, underground water simulation, the black oil model in reservoir simulation, and so on.

Install

To compile, you need a C99 compiler (and a F90 compiler if you need Fortran examples). By default, we use GNU gcc/gfortan, respectively.

Configuring and building the FASP library and test suite requires CMake 2.8 or higher http://www.cmake.org/.

The command to configure is:

```
$ mkdir Build; cd Build; cmake ..
```

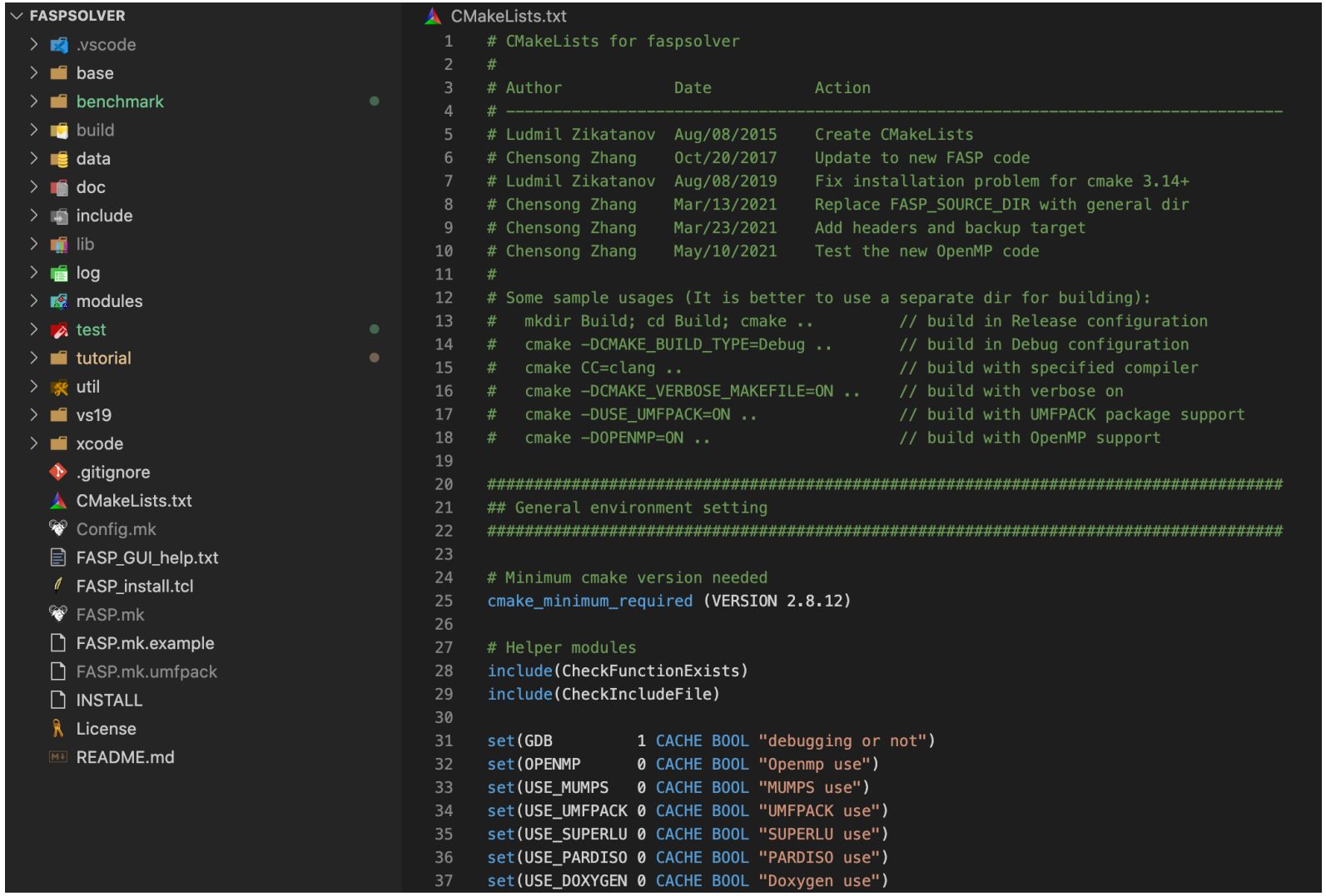
After successfully configing the environment, just run:

```
$ make    // to compile the FASP static library
```

To install the FASP library and executables, run:

```
$ make install
```

faspsolver安装选项介绍



faspsolver简单算例（C语言）

```c
/*! \file  poisson-pcg.c
 *
 *  \brief The third test example for FASP: using PCG to solve
 *         the discrete Poisson equation from P1 finite element.
 *         C version.
 *
 *  \note  Solving the Poisson equation (P1 FEM) with PCG: C version
 *
 *--------------------------------------------------------------------------------
 *  Copyright (C) 2012--Present by the FASP team. All rights reserved.
 *  Released under the terms of the GNU Lesser General Public License 3.0 or later.
 *--------------------------------------------------------------------------------
 */

#include "fasp.h"
#include "fasp_functs.h"

/**
 * \fn int main (int argc, const char * argv[])
 *
 * \brief This is the main function for the third example.
 *
 * \author Feiteng Huang
 * \date   05/17/2012
 *
 * Modified by Chensong Zhang on 09/22/2012
 * Modified by Chensong Zhang on 12/23/2018: Fix memory leakage
 */
int main (int argc, const char * argv[])
{
    input_param     inparam;  // parameters from input files
    ITS_param       itparam;  // parameters for itsolver
    AMG_param       amgparam; // parameters for AMG
    ILU_param       iluparam; // parameters for ILU

    printf("\n====================================");
    printf("\n||   FASP: PCG example -- C version   ||");
    printf("\n====================================\n\n");

    // Step 0. Set parameters: We can use ini/pcg.dat
    fasp_param_set(argc, argv, &inparam);
    fasp_param_init(&inparam, &itparam, &amgparam, &iluparam, NULL);

    // Set local parameters
    const SHORT print_level = itparam.print_level;
    const SHORT pc_type     = itparam.precond_type;
    const SHORT stop_type   = itparam.stop_type;
    const INT   maxit       = itparam.maxit;
    const REAL  tol         = itparam.tol;

    // Step 1. Get stiffness matrix and right-hand side
    // Read A and b -- P1 FE discretization for Poisson. The location
    // of the data files is given in "ini/pcg.dat".
    dCSRmat A;
    dvector b, x;
    char filename1[512], *datafile1;
    char filename2[512], *datafile2;

    // Read the stiffness matrix from matFE.dat
    memcpy(filename1, inparam.workdir, STRLEN);
    datafile1="csrmat_FE.dat"; strcat(filename1, datafile1);

    // Read the RHS from rhsFE.dat
    memcpy(filename2, inparam.workdir, STRLEN);
    datafile2="rhs_FE.dat"; strcat(filename2, datafile2);

    fasp_dcsrvec_read2(filename1, filename2, &A, &b);

    // Step 2. Print problem size and PCG parameters
    if (print_level>PRINT_NONE) {
        printf("A: m = %d, n = %d, nnz = %d\n", A.row, A.col, A.nnz);
        printf("b: n = %d\n", b.row);
        fasp_param_solver_print(&itparam);
    }

    // Setp 3. Setup preconditioner
    // Preconditioner type is determined by pc_type
    precond *pc = fasp_precond_setup(pc_type, &amgparam, &iluparam, &A);

    // Step 4. Solve the system with PCG as an iterative solver
    // Set the initial guess to be zero and then solve it using PCG solver
    // Note that we call PCG interface directly. There is another way which
    // calls the abstract iterative method interface; see possion-its.c for
    // more details.
    fasp_dvec_alloc(A.row, &x);
    fasp_dvec_set(A.row, &x, 0.0);

    fasp_solver_dcsr_pcg(&A, &b, &x, pc, tol, maxit, stop_type, print_level);

    // Step 5. Clean up memory
    // First clean up the AMG data if you are using AMG as preconditioner
    fasp_amg_data_free(((precond_data *)pc->data)->mgl_data, &amgparam);

    // Clean up the pcdata
    if (pc_type!=PREC_NULL) fasp_mem_free(pc->data);
    fasp_mem_free(pc);

    // Clean up coefficient matrix, right-hand side, and solution
    fasp_dcsr_free(&A);
    fasp_dvec_free(&b);
    fasp_dvec_free(&x);

    return FASP_SUCCESS;
}
```

faspsolver简单算例（Fortran语言）

```
!> \file  poisson-pcg.f90
!>
!> \brief The third test example for FASP: using PCG to solve
!>        the discrete Poisson equation from P1 finite element.
!>        F90 version.
!>
!> \note  Solving the Poisson equation (P1 FEM) with PCG: F90 version
!>
!>--------------------------------------------------------------------------------
!>  Copyright (C) 2012--Present by the FASP team. All rights reserved.
!>  Released under the terms of the GNU Lesser General Public License 3.0 or later.
!>--------------------------------------------------------------------------------

program test

  implicit none

  double precision, dimension(:), allocatable :: u,b
  double precision, dimension(:), allocatable :: a
  integer,          dimension(:), allocatable :: ia,ja

  integer          :: iufile, n, nnz, i, prt_lvl, maxit
  double precision :: tol

  print*, ""
  write(*,"(A)") "========================================="
  write(*,"(A)") "||   FASP: PCG example -- F90 version   ||"
  write(*,"(A)") "========================================="
  print*, ""

  ! Step 0: user defined variables
  prt_lvl = 3
  maxit = 500
  tol = 1.0d-6
  iufile = 1

  ! Step 1: read A and b

  !===> Read data A from file
  open(unit=iufile,file='../data/csrmat_FE.dat')

  read(iufile,*) n
  allocate(ia(1:n+1))
  read(iufile,*) (ia(i),i=1,n+1)

  nnz=ia(n+1)-ia(1)
  allocate(ja(1:nnz),a(1:nnz))
  read(iufile,*) (ja(i),i=1,nnz)
  read(iufile,*) (a(i),i=1,nnz)

  close(iufile)

  !===> Read data b from file
  open(unit=iufile,file='../data/rhs_FE.dat')

  read(iufile,*) n
  allocate(b(1:n))
  read(iufile,*) (b(i),i=1,n)

  close(iufile)

  !===> Shift the index to start from 0 (for C routines)
  forall (i=1:n+1) ia(i)=ia(i)-1
  forall (i=1:nnz) ja(i)=ja(i)-1

  ! Step 2: Solve the system

  !===> Initial guess
  allocate(u(1:n))
  u=0.0d0
  call fasp_fwrapper_dcsr_krylov_amg(n,nnz,ia,ja,a,b,u,tol,maxit,prt_lvl);

  ! Step 3: Clean up memory
  deallocate(ia,ja,a)
  deallocate(b,u)

end program test
```

faspsolver算法参数配置

```
%-------------------------------------------%
% parameters for multilevel iteration       %
%-------------------------------------------%

AMG_type                 = C      % C classic AMG
                                  % SA smoothed aggregation
                                  % UA unsmoothed aggregation
AMG_cycle_type           = V      % V V-cycle | W W-cycle
                                  % A AMLI-cycle | NA Nonlinear AMLI-cycleA
AMG_tol                  = 1e-8   % tolerance for AMG
AMG_maxit                = 100    % number of AMG iterations
AMG_levels               = 20     % max number of levels
AMG_coarse_dof           = 500    % max number of coarse degrees of freedom
AMG_coarse_scaling       = OFF    % switch of scaling of the coarse grid correction
AMG_amli_degree          = 2      % degree of the polynomial used by AMLI cycle
AMG_nl_amli_krylov_type  = 6      % Krylov method in NLAMLI cycle: 6 FGMRES | 7 GCG

%-------------------------------------------%
% parameters for AMG smoothing              %
%-------------------------------------------%

AMG_smoother             = GS     % GS | JACOBI | SGS
                                  % SOR | SSOR | GSOR | SGSOR | POLY
AMG_ILU_levels           = 0      % number of levels using ILU smoother
AMG_SWZ_levels           = 0      % number of levels using Schwarz smoother
AMG_relaxation           = 1.1    % relaxation parameter for SOR smoother
AMG_polynomial_degree    = 3      % degree of the polynomial smoother
AMG_presmooth_iter       = 2      % number of presmoothing sweeps
AMG_postsmooth_iter      = 2      % number of postsmoothing sweeps

%-------------------------------------------%
% parameters for classical AMG SETUP        %
%-------------------------------------------%

AMG_coarsening_type      = 1      % 1 Modified RS
                                  % 3 Compatible Relaxation
                                  % 4 Aggressive
AMG_interpolation_type   = 1      % 1 Direct | 2 Standard | 3 Energy-min
AMG_strong_threshold     = 0.6    % Strong threshold
AMG_truncation_threshold = 0.4    % Truncation threshold
AMG_max_row_sum          = 0.9    % Max row sum
```

Problem	DOF	RS-V-CG	UA-NA-CG	hypre	AGMG
2D 5pt	1M	1.79	1.43	1.96	1.73
2D 5pt	4M	8.71	6.04	8.45	6.61
2D 9pt	1M	1.82	2.07	2.25	2.24
2D 9pt	4M	7.63	8.39	8.88	9.42
3D 7pt	$\frac{1}{4}$M	1.05	0.37	1.83	0.43
3D 7pt	2M	10.86	3.28	19.04	3.71
3D 27pt	$\frac{1}{4}$M	2.09	0.94	3.26	1.79
3D 27pt	2M	20.0	8.53	34.54	20.29

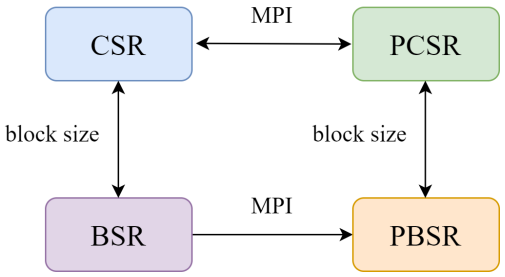
Test Device: Intel Core i5 2.6GHz, 8GB RAM, gcc 4.9.2 -O2. Software ver: FASP 1.7.0, hypre 2.10.0b, AGMG 3.2.0 (default parameters). Computing time (seconds) of the AMG-preconditioned CG method. We solve the 2D/3D Poisson equation with one processing core. Stopping criteria: relative residual is less than 1E-6.

faspxx并行解法器软件简介

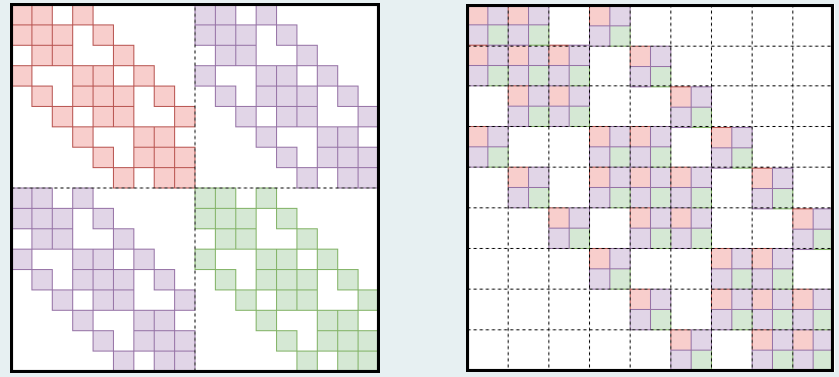
- A flexible and extensible package of iterative solvers/preconditioners: https://gitee.com/faspdevteam/faspxx
- Features: Parallel (MPI/OpenMP) support; a unified design for solver/precond/smoother design, KSM: CG, BiCGStab, GMRES(m), FGMRES-DR, …; controls via a JSON file

Fast Auxiliary Subspace Preconditioned, fleXible and eXtensible solvers (faspxx)

应用	Poisson	Petroleum Reservoir Simulation	Elasticity	...
算法	Preconditioners: AMG, BAMG, ILU, BILU Krylov methods: CG, GMRES, FGMRES, FGMRES-DR, BiCGStab BCG, BGMRES, BFGMRES,BBiCGStab			
组件	MAT	ITM	LS	
底层	CSR, BSR, PCSR, PBSR	MPI, OpenMP	JSON	



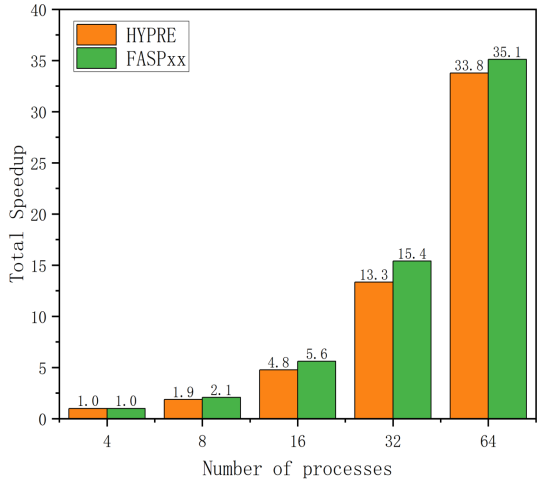
Matrix Format

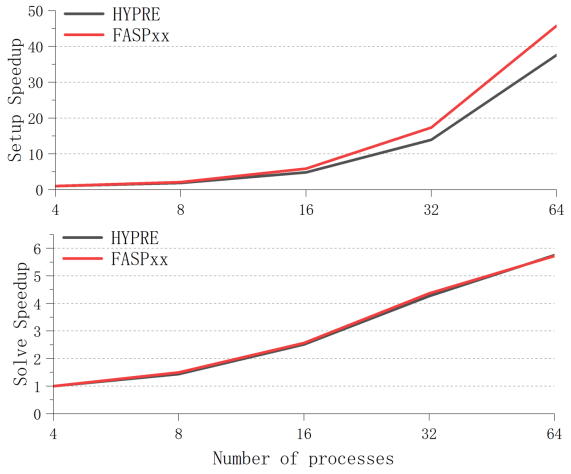


faspxx的并行性能测试（C-AMG解法器）

Test 1. FEM for Poisson, grid: 2048x2048, dof: 4M, AMG-CG

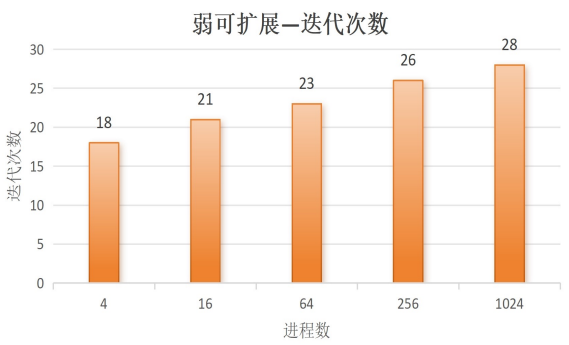
	BoomerAMG (default, OC=2.24)				FASPxx CAMG (OC=1.88)			
NP	IT	Setup (s)	Solve (s)	Total (s)	IT	Setup (s)	Solve (s)	Total (s)
4	14	313.97	6.48	320.44	18	180.09	8.16	188.25
8	15	166.90	4.50	171.41	18	85.45	5.43	90.88
16	15	64.76	2.58	67.34	18	30.43	3.18	33.60
32	15	22.50	1.52	24.02	18	10.36	1.87	12.23
64	15	8.36	1.13	9.48	18	3.93	1.43	5.36

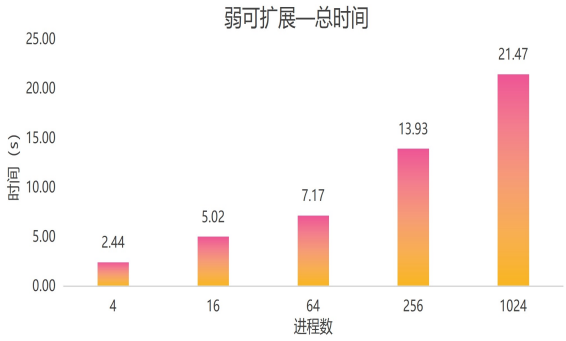




Test 2. FEM for Poisson, dof/np: 64,000, AMG-CG

FASPxx CAMG (OC=1.72)					
NP	Mesh	IT	Setup (s)	Solve (s)	Total (s)
4	512x512	18	2.00	0.43	2.44
16	1024x1024	21	4.35	0.67	5.02
64	2048x2048	23	5.79	1.38	7.17
256	4096x4096	26	7.58	6.36	13.93
1024	8192x8192	28	10.29	11.18	21.47





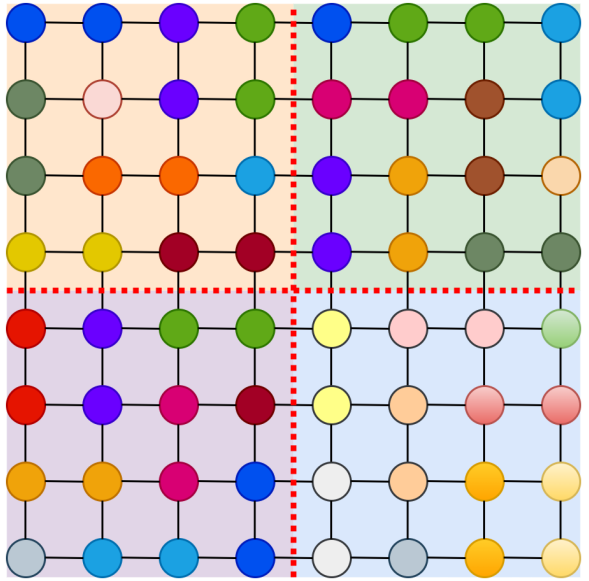
Strong and weak scalability performance test for the discrete Poisson's equation (finite element)

faspxx的并行性能测试（A-AMG解法器）

Test 3. FDM for Poisson, grid: 4096x4096, dof: 16M, AMG-CG

	AG-AMG (OC=1.14)				FASPxx UA-AMG (OC=1.14)			
NP	IT	Setup (s)	Solve (s)	Total (s)	IT	Setup (s)	Solve (s)	Total (s)
1	50	5.69	25.50	31.19	48	4.63	33.27	37.90
2	50	2.76	13.20	15.96	51	2.52	19.58	22.10
4	50	1.40	7.10	8.50	82	1.59	15.27	16.86
8	50	0.74	3.43	4.17	80	0.85	7.60	8.45
16	82	0.44	3.17	3.61	80	0.51	4.34	4.85
32	85	0.34	2.34	2.68	80	0.39	2.84	3.22

Parallel Pairwise aggregation method

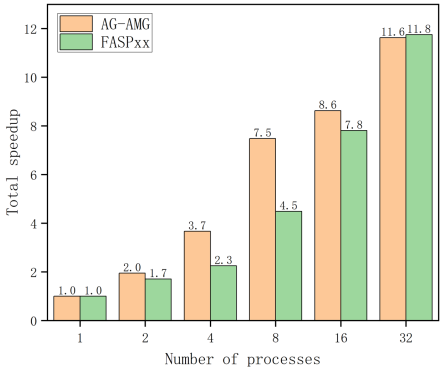


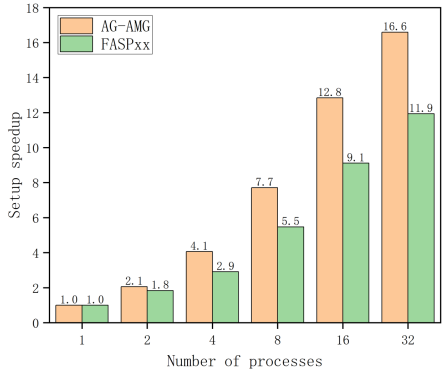
- Tentative prolongation

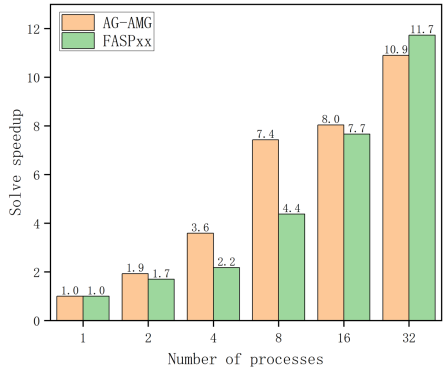
$$P_{ij} := \begin{cases} 1, & i \in \mathcal{A}_j \\ 0, & i \notin \mathcal{A}_j \end{cases}$$

- Coarser matrix

$$A_c := RAP = P^T AP$$







For the discrete Poisson's equation (finite differenece), strong scalability performance of the faspxx solver is similar to that of AGMG

线性解法器不可能三角

Q: 如何改进Multigrid方法实现？目标？计算资源？用户类型？开发代价？有无高效算法?

高效性

Efficient, optimal, and scalable

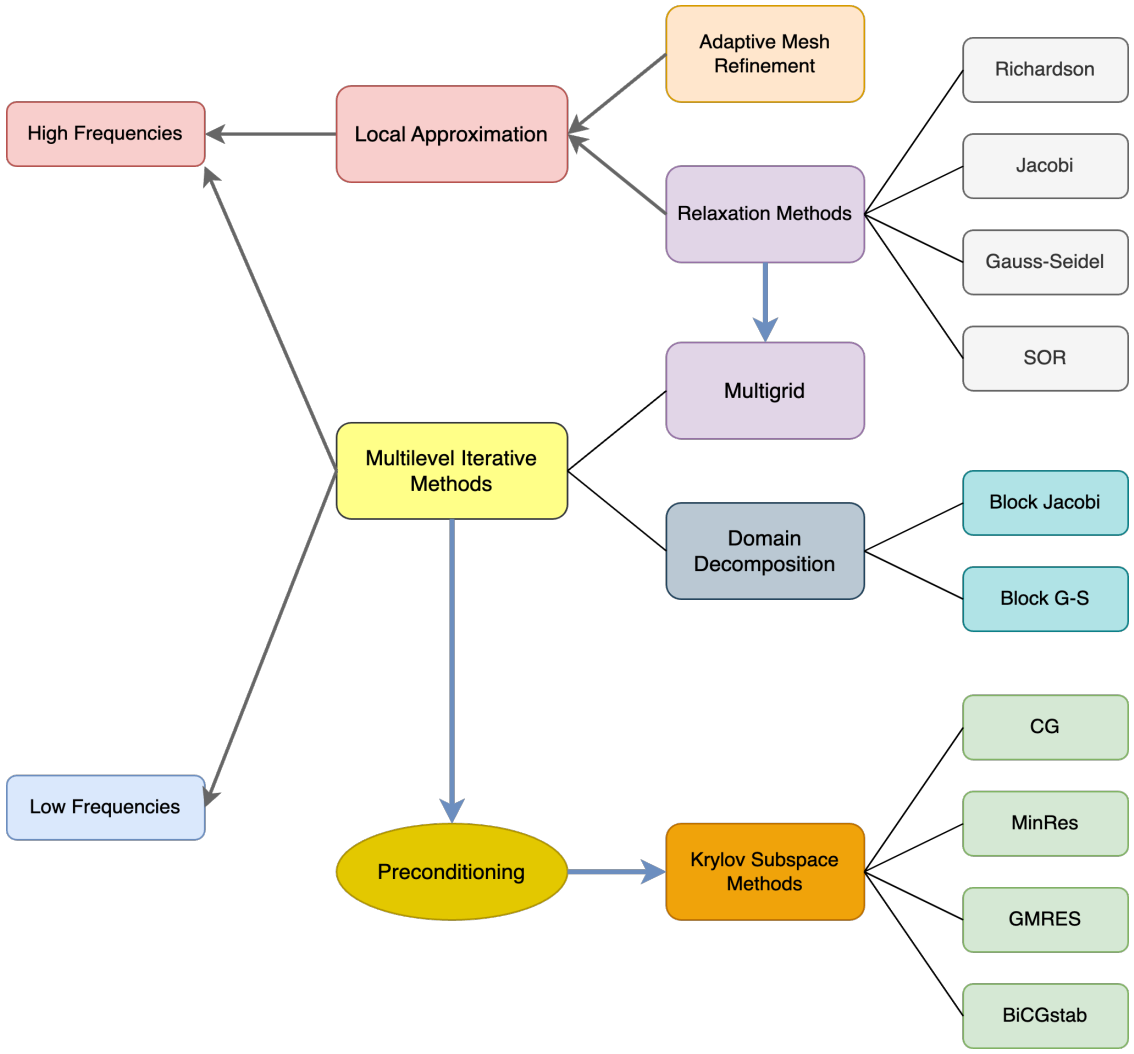
稳健性

Robust, resilient, and reliable

易用性

Applicable, user-friendly, and portable

如何选择求解方法



- 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
- Iain Duff: "*Real men use direct methods*"
- Use direct methods whenever possible (only for moderate-size problems)
- **Think about your solution method as early as possible for your simulation**

解法器课程与竞赛


端到端时间可分为如下3部分:
1. 分析
2. 准备
3. 求解
矩阵和向量的读入、格式转换、分发至多进程、矩阵坐标排序不列入统计时间。
直接法和迭代法对于上述3步的定义略有不同。
做解法器预处理，包括图分析、图优化、
iso的phase=11（注意，pardiso phase=11
。
pardiso phase=22。
pardiso phase=33。
竞赛说明
中国科学院大学
夏季强化课程
2022
Fast Solvers for
Large Algebraic Systems
Brief Introduction on This Course
Chensong Zhang, A
快速解法器课程
赛题名称
solverchallenge23_01
阶数非零元数
2,097,152
非零元数
14,581,760
应用领域
激光聚变
来源与背景描述
MD5
约束
矩阵说明
赛题简介
竞赛方案与规则
竞赛方案
竞赛组委会提供10套来源于不同应用领域的线性系统（每套线性系统包括矩阵A和右端向量b、存储格式的描述、求解约束条件等），参赛选手针对这10套系统进行求解，不限制求解方法（即迭代法、直接法或混合方法均可），也不限制解法器软件包的来源。竞赛网站将提供程序中输入、输出与计时部分，并开放接口供参赛选手提交自己的执行代码（若是闭源的，则提交链接文件）。最终将使用不同的右端向量b测试，其求解效果（速度、精度、浮点运算次数等）将作为本竞赛奖项评判标准。
竞赛规则
竞赛方案与规则

Chensong Zhang, AMSS

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

Release version 2024.11.25