Lecture on Solving Large Matrix Problems Effectively

Matthias Bollhöfer

TU Braunschweig, Inst. *Computational Mathematics*

University of Warwick, November 24, 2011

Technische Universität Braunschweig

Outline

Outline

Model Problems

Where do large matrix problems show up?

1D elliptic boundary value problem

$$\begin{aligned} -u''(x) &= f(x),\ x \in [0,1] \\ u(0) &= g_0,\ u(1) = g_1 \end{aligned}$$

2D elliptic boundary value problem

$$\begin{aligned} \Omega &= [0,1] \times [0,1] \\ \overbrace{-u_{xx}(x,y) - u_{yy}(x,y)}^{-\Delta u(x,y)} &= f(x,y),\ (x,y) \in \Omega \\ u(x,y) &= g(x,y),\ (x,y) \in \partial\Omega \end{aligned}$$

3D elliptic boundary value problem

$$-\Delta u = f \text{ in } [0,1]^3 + \text{b.c.}$$

Discretization using Finite Differences

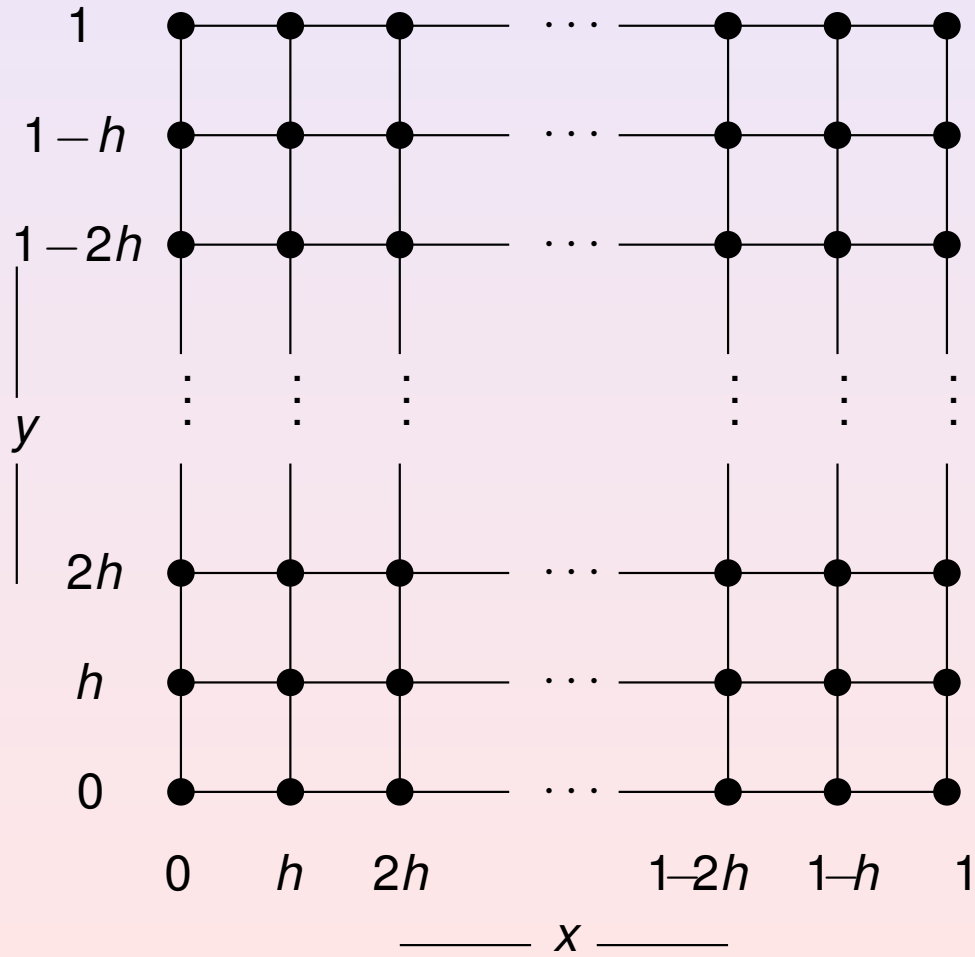
Where do large matrix problems show up?

1D problem: $h = \frac{1}{N+1}$. $\quad [0,1] \to \Omega_h = \{0, h, 2h, \ldots, 1-h, 1\}$

2D problem:

$\Omega = [0,1]^2 \to \Omega_h =$

$\{(k,l)h : \; k,l = 0, \ldots N+1\}$



Discretization using Finite Differences in 1D

Where do large matrix problems show up?

1D boundary value problem

Differential equation

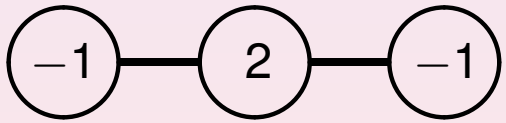
$$-u''(x) = f(x),\ u(0) = g_0, u(1) = g_1$$

$$\downarrow$$

Difference equation

$$\frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} = f_i, u_0 = g_0, u_{N+1} = g_1, \text{ where } u_i \equiv u(ih)\ \forall i$$

3-point stencil



Linear System

Where do large matrix problems show up?

$$T_h = \frac{1}{h^2} \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{pmatrix} \longrightarrow T_h u = f.$$

Discretization using Finite Differences in 2D

Where do large matrix problems show up?

2D boundary value problem

Differential equation

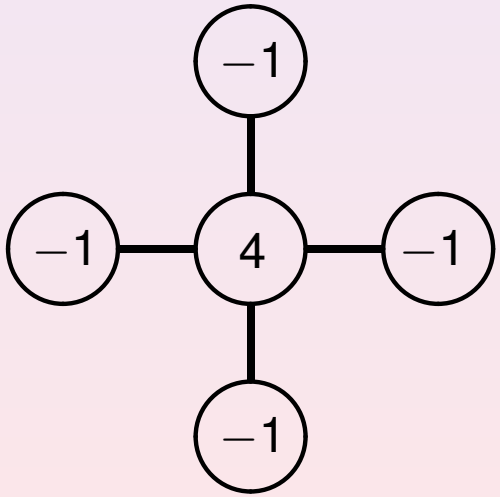
$$-u_{xx}(x,y) - u_{yy}(x,y) \equiv -\Delta u(x,y) = f(x,y), \text{ in } \Omega = [0,1]^2, \; y = g \text{ on } \partial\Omega$$

$$\downarrow$$

Difference equation

$$\frac{-u_{i-1,j} - u_{i,j-1} + 4u_{ij} - u_{i+1,j} - u_{i,j+1}}{h^2} = f_{ij}$$

5-point-difference stencil



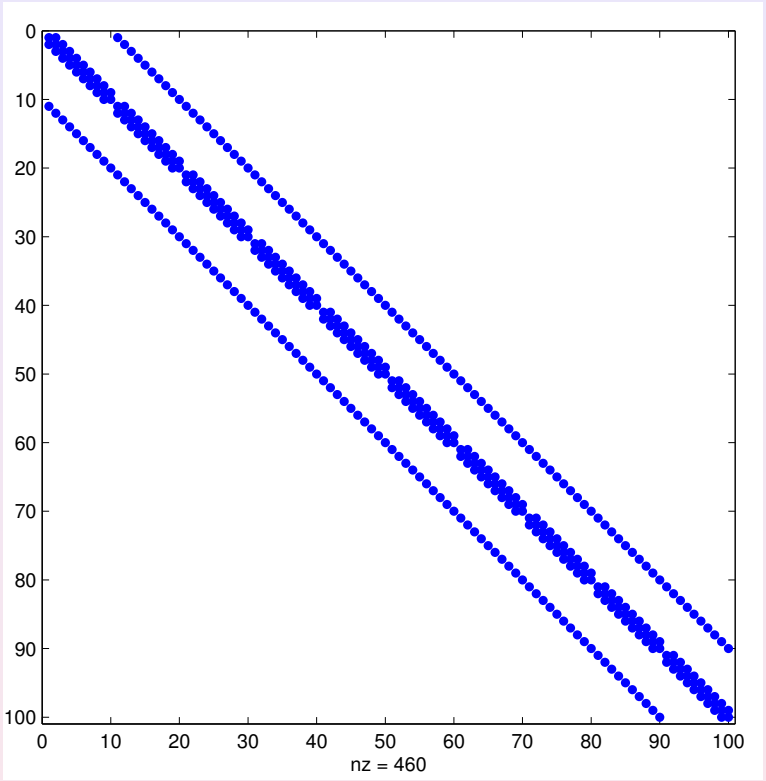
Linear system

$$A_h u = f$$

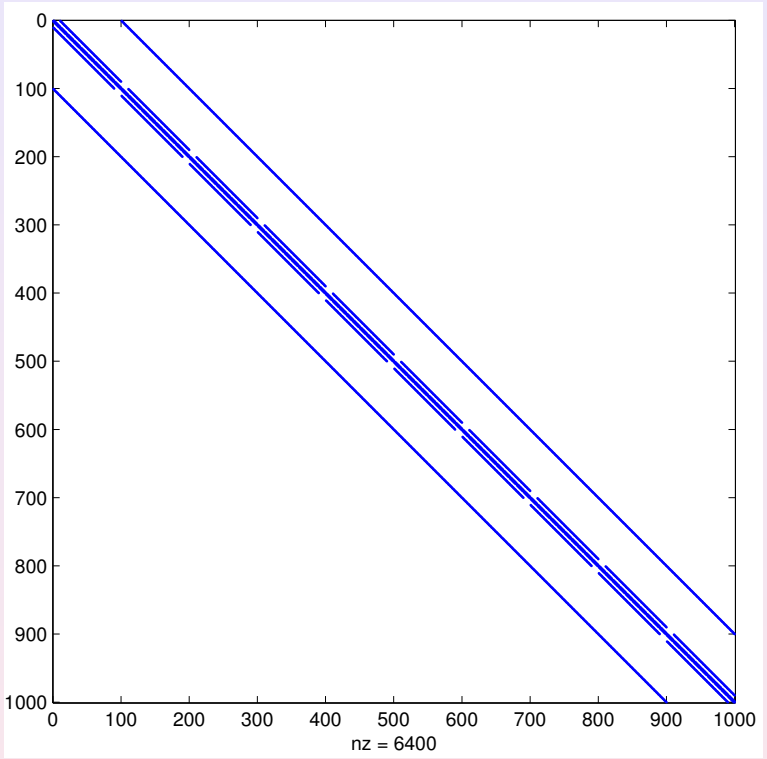
Large Sparse Linear Systems

Where do large matrix problems show up?

2D



3D

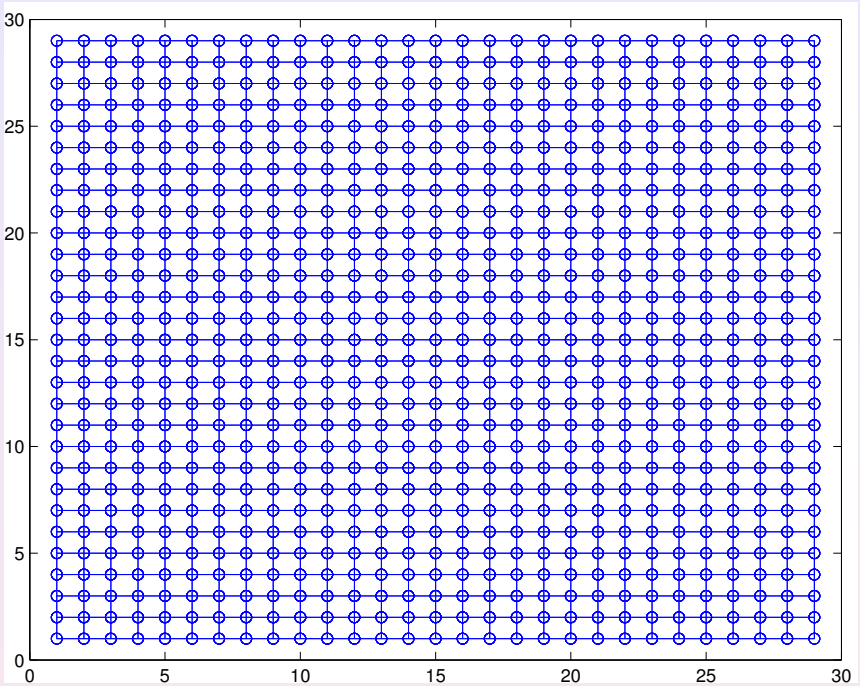


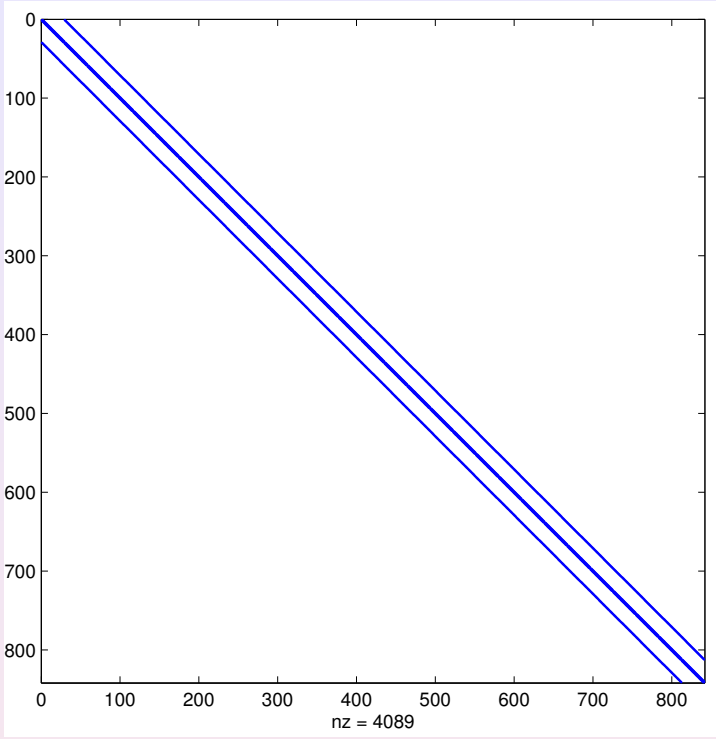
- discretization of partial differential equations leads to large sparse systems
- size of the matrix increases as $h \to 0$ but also depends on the spatial dimension.
- nonzero pattern of a matrix A is connected to the discretization of the domain, $G_A := (V, E)$, $V = \{1, \ldots, n\}$, $E = \{(p, q) : \ a_{pq} \neq 0\}$, "graph of a matrix"

Outline

Partitioning the System

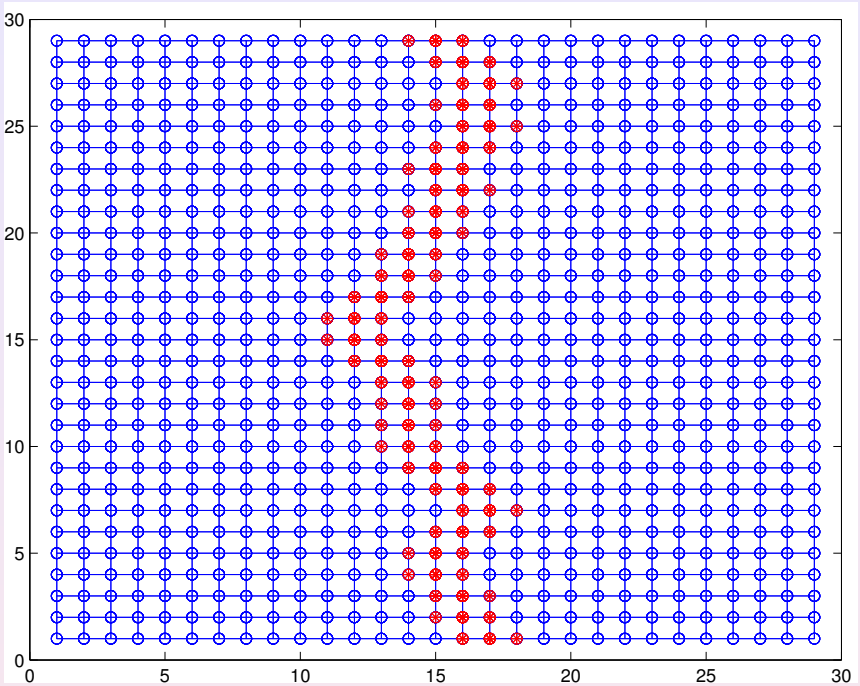
Nested Dissection Preprocessing Step

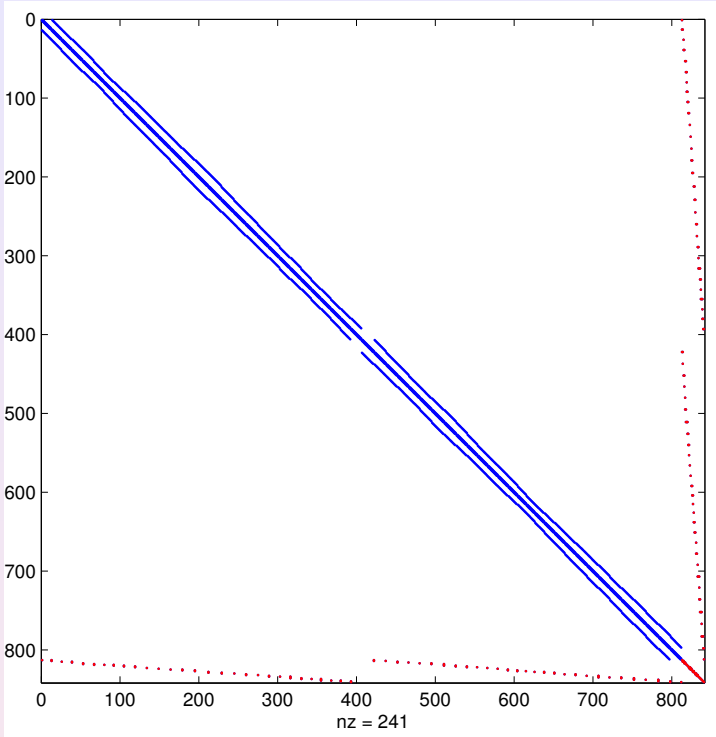


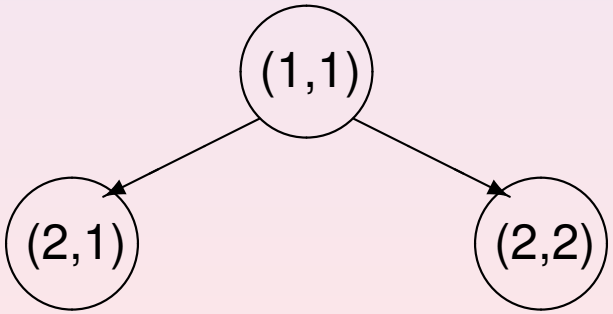


Partitioning the System

Nested Dissection Preprocessing Step

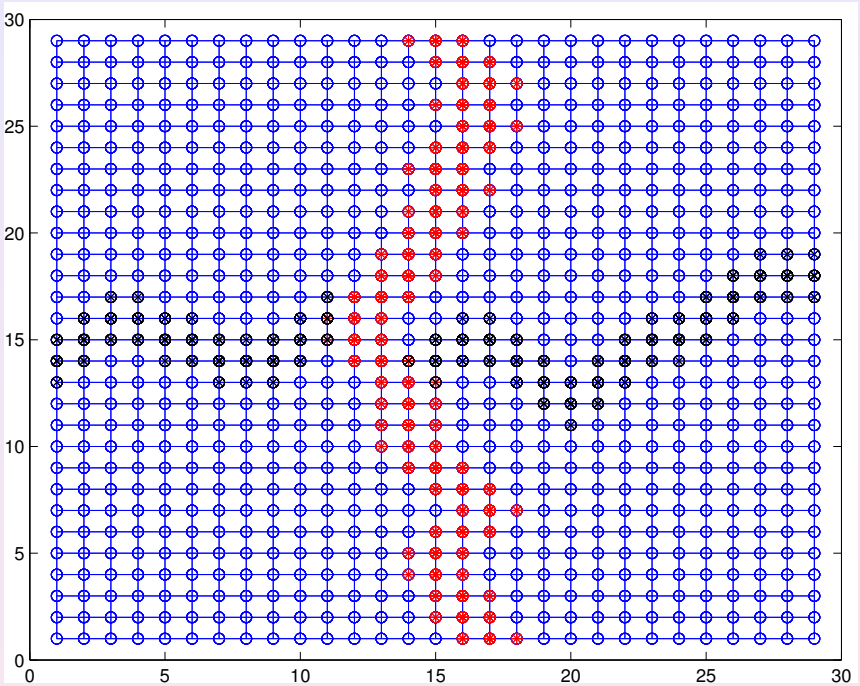


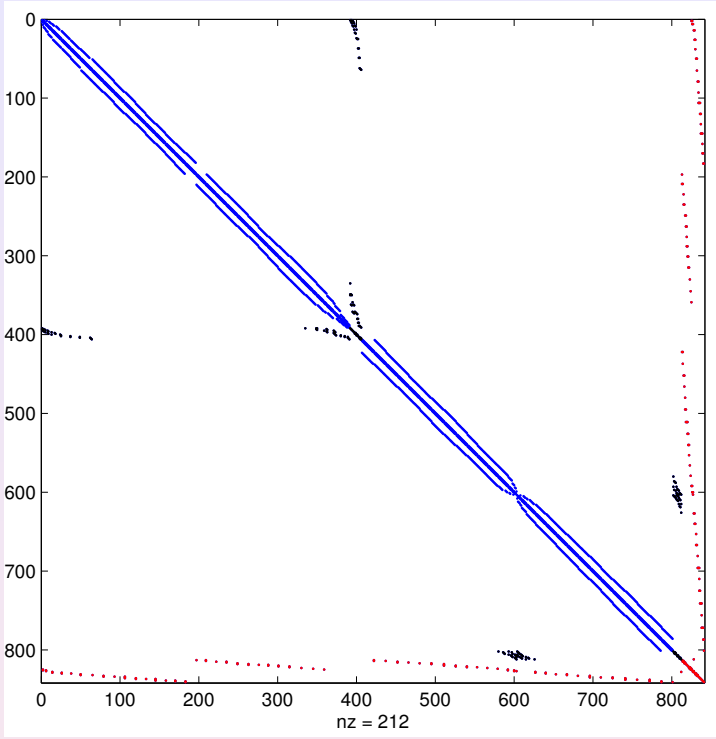


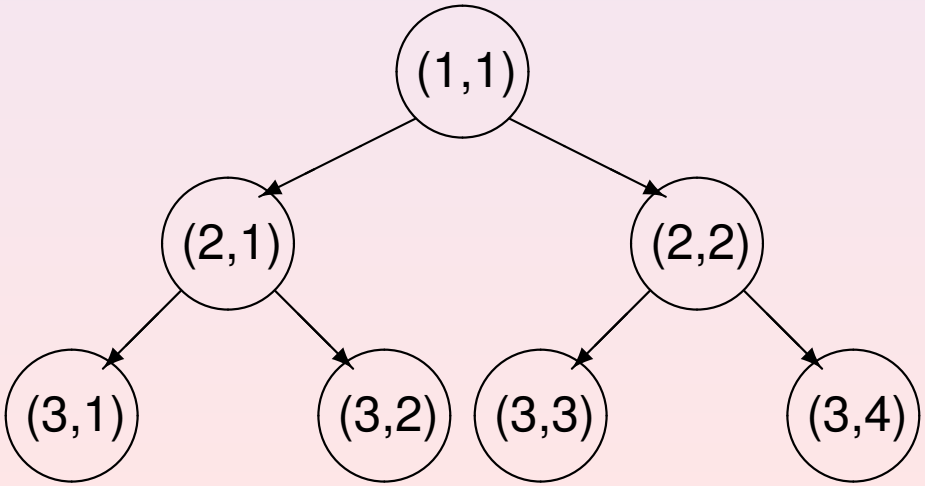


Partitioning the System

Nested Dissection Preprocessing Step







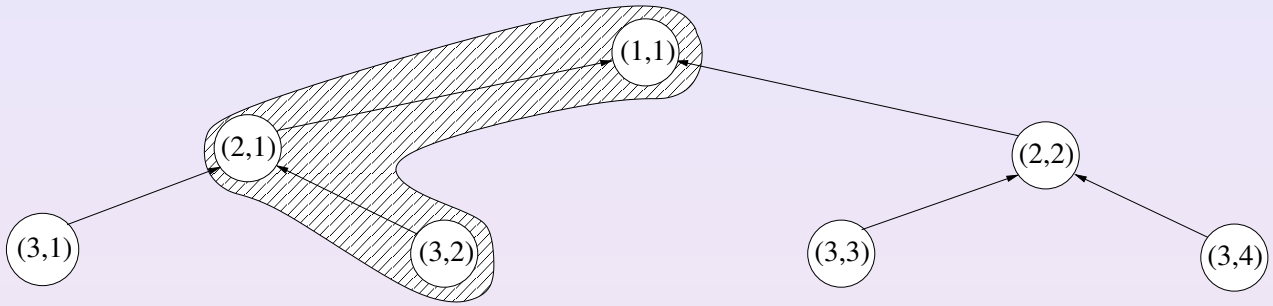
Partitioning the System

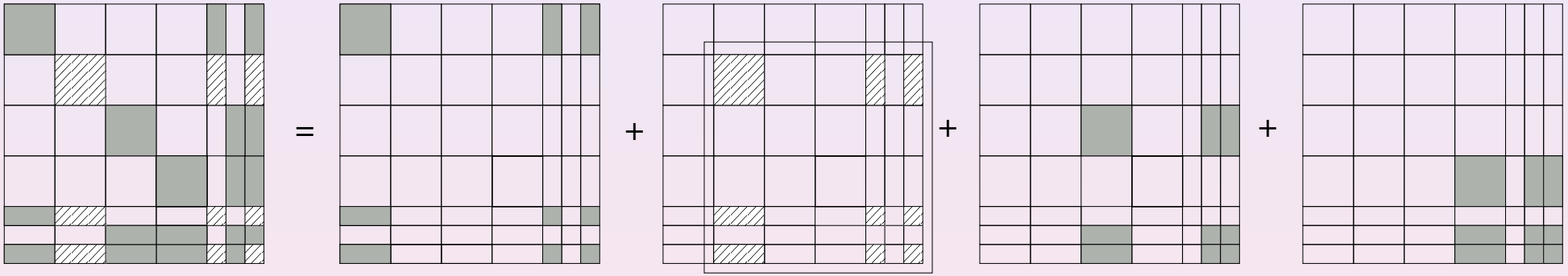
Nested Dissection Preprocessing Step

- each nested dissection step creates two decoupled subproblems
- the more subproblems we produce (such as the number of processors/threads), the more couplings we obtain
- popular nested dissection codes METIS (Karypis/Kumar), SCOTCH (Pellegrini)
- parallel version of METIS (PARMETIS) and SCOTCH (PT-SCOTCH) based on MPI are available

Parallel System Splitting

Parallel Matrix Splitting





$\underline{A} = \underline{A}^{(3,1)} + \underline{A}^{(3,2)} + \underline{A}^{(3,3)} + \underline{A}^{(3,4)}$

Parallel Vector Splitting

$\underline{x} = \underline{x}^{(3,1)} + \underline{x}^{(3,2)} + \underline{x}^{(3,3)} + \underline{x}^{(3,4)}$

Outline

Iterative Solution

$$Ax = b$$

- To simplify the discussion, suppose that A is symmetric positive definite (SPD).
- In modern Numerical Linear Algebra the default iterative solver for SPD systems is the CG method (conjugate gradients)

$$f(x) := \frac{1}{2}x^T Ax - x^T b \text{ convex function}$$

$$f(x) \overset{!}{=} \min \Rightarrow \nabla f(x) = 0 \Leftrightarrow Ax - b = 0$$

Objective: provide search directions $t_1, t_2, t_3, \ldots$ and successively minimize f along the search directions

0. x_0 initial guess.

1a. compute α_1 s.t. $g_1(\alpha_1) := f(x_0 + \alpha_1 t_1)$ is minimal

1b. $x_1 := x_0 + \alpha_1 t_1$

2a. compute α_2 s.t. $g_2(\alpha_2) := f(x_1 + \alpha_2 t_2)$ is minimal

2b. $x_2 := x_1 + \alpha_2 t_2$

$\vdots$

Iterative Solution

- each α_k is easily computed by elementary calculus as

$$\alpha_k = \frac{t_k^T r_k}{t_k^T t_k}, \text{ where } r_k = b - Ax_k \text{ residual}$$

- locally a natural choice for t_k would be the negative gradient (steepest descent)

$$t_k = -\nabla f(x_{k-1} + \alpha_k t_k) = r_k.$$

  on a global scale this is BAD.
- globally optimal are A-conjugate search directions $t_1, t_2, t_3, \ldots$

$$t_k^T A t_l = 0, \text{ for all } k \neq l$$

  This choice ensures that $\alpha_1, \ldots, \alpha_k$ also satisfy the k-dimensional minimization problem

$$\min_{\alpha_1,\ldots,\alpha_k} h(\alpha_1, \ldots, \alpha_k), \text{ where } h(\alpha_1, \ldots, \alpha_k) := f(x_0 + \alpha_1 t_1 + \cdots + \alpha_k t_k).$$

- the associated method is called CG (conjugate gradients)

The CG Method

x initial guess
$r = b - Ax$
$t = r$
$\rho = r^T r$
for $k = 1, 2, 3, \ldots$
  $z = At$
  $\alpha = \rho / (t^T z)$
  $x = x + \alpha t$
  $r = r - \alpha z$
  $\rho_{old} = \rho$
  $\rho = r^T r$
  $\beta = \rho / \rho_{old}$
  $t = r + \beta t$
end

Hiding the technical details of the derivation, CG requires the computation of:

- matrix-vector products
- scalar products
- vector updates

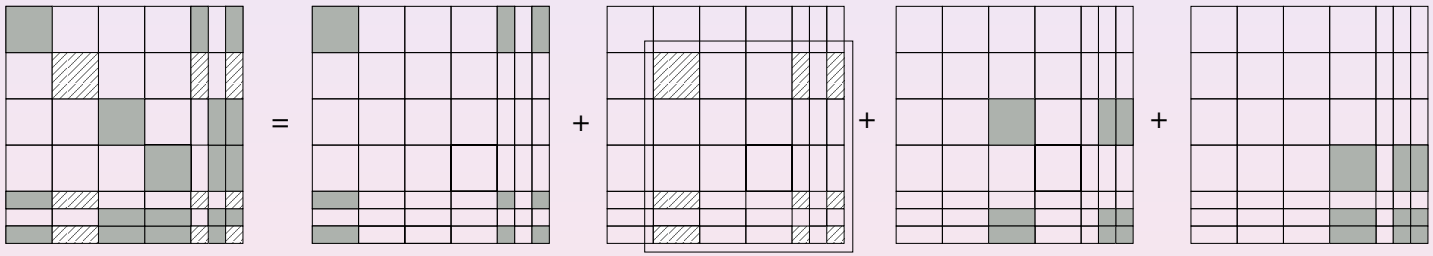
Outline

The Parallel Matrix-Vector Multiplication

Suppose that $p = 2^{s-1}$ and A is split as

$$\underline{A} = \underline{A}^{(s,1)} + \underline{A}^{(s,2)} + \cdots + \underline{A}^{(s,p)}$$

using nested dissection (binary tree, see earlier slides).



The single matrices $\underline{A}^{(s,q)}$ are distributed over the processors (MPI case) or accessed by different threads (OpenMP)

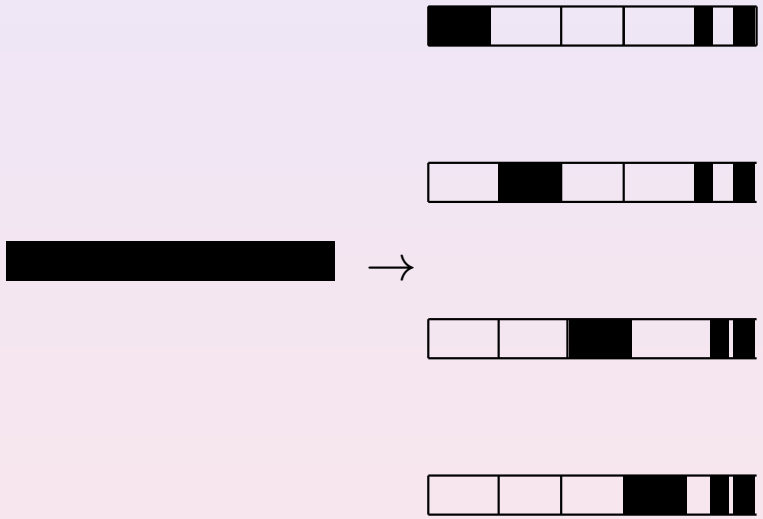
The Parallel Matrix-Vector Multiplication

Consider two classes of vectors

- vectors with **consistent** representation
- vectors with inconsistent representation

1. **consistent** vector copies

$$\mathbf{x} \rightarrow \begin{cases} \mathbf{x}^{(s,1)} \\ \mathbf{x}^{(s,2)} \\ \vdots \\ \mathbf{x}^{(s,p)} \end{cases}$$



2. inconsistent vector splitting

$$\underline{r} = \underline{r}^{(s,1)} + \underline{r}^{(s,1)} + \cdots + \underline{r}^{(s,p)}$$

Parallel Matrix-Vector Multiplication

1. suppose that A is split as $\underline{A} = \underline{A}^{(s,1)} + \underline{A}^{(s,2)} + \cdots + \underline{A}^{(s,p)}$
2. Assume that $\mathbf{t}$ is represented **consistently** by $\mathbf{t}^{(s,1)}, \mathbf{t}^{(s,2)}, \ldots, \mathbf{t}^{(s,p)}$

- matrix-vector multiplication can be performed in parallel

$$\Rightarrow \underline{A}\mathbf{t} = (\sum_{q=1}^{p} \underline{A}^{(s,q)})\mathbf{t} = \sum_{q=1}^{p} (\underline{A}^{(s,q)}\mathbf{t}^{(s,q)})$$

- define $\underline{z}^{(s,q)}$ by

$$\underline{z}^{(s,q)} := \underline{A}^{(s,q)}\mathbf{t}^{(s,q)} \quad \Longrightarrow \quad \underline{z} = \underline{z}^{(s,1)} + \underline{z}^{(s,2)} + \cdots + \underline{z}^{(s,q)}$$

Parallel Matrix-Vector Multiplication

1. suppose that A is split as $\underline{A} = \underline{A}^{(s,1)} + \underline{A}^{(s,2)} + \cdots + \underline{A}^{(s,p)}$
2. Assume that $\mathbf{t}$ is represented **consistently** by $\mathbf{t}^{(s,1)}, \mathbf{t}^{(s,2)}, \ldots, \mathbf{t}^{(s,p)}$

- matrix-vector multiplication can be performed in parallel

$$\Rightarrow \underline{A}\mathbf{t} = (\sum_{q=1}^{p} \underline{A}^{(s,q)})\mathbf{t} = \sum_{q=1}^{p} (\underline{A}^{(s,q)}\mathbf{t}^{(s,q)})$$

- define $\underline{z}^{(s,q)}$ by

$$\underline{z}^{(s,q)} := \underline{A}^{(s,q)}\mathbf{t}^{(s,q)} \quad \Longrightarrow \quad \underline{z} = \underline{z}^{(s,1)} + \underline{z}^{(s,2)} + \cdots + \underline{z}^{(s,q)}$$

Observation

1. format of the matrix-vector multiplication becomes an inconsistent representation automatically
2. no communication required

Parallel Scalar Product

scalar products can be performed easily if the vectors are of different type

Parallel Scalar Product

scalar products can be performed easily if the vectors are of different type

Let **t** be **consistent** and $\underline{z}$ be inconsistent

$$\sigma = \mathbf{t}^T \underline{z} = \mathbf{t}^T \left(\sum_{q=1}^{p} \underline{z}^{(s,q)}\right) = \sum_{q=1}^{p} (\mathbf{t}^{(s,q)})^T \underline{z}^{(s,q)}.$$

Parallel Scalar Product

scalar products can be performed easily if the vectors are of different type

Let **t** be **consistent** and $\underline{z}$ be inconsistent

$$\sigma = \mathbf{t}^T \underline{z} = \mathbf{t}^T \left(\sum_{q=1}^{p} \underline{z}^{(s,q)}\right) = \sum_{q=1}^{p} (\mathbf{t}^{(s,q)})^T \underline{z}^{(s,q)}.$$

Local scalar products $\sigma_q := (\mathbf{t}^{(s,q)})^T \underline{z}^{(s,q)}$ can be computed in parallel

Parallel Scalar Product

scalar products can be performed easily if the vectors are of different type

Let **t** be **consistent** and $\underline{z}$ be inconsistent

$$\sigma = \mathbf{t}^T \underline{z} = \mathbf{t}^T \left(\sum_{q=1}^{p} \underline{z}^{(s,q)}\right) = \sum_{q=1}^{p} (\mathbf{t}^{(s,q)})^T \underline{z}^{(s,q)}.$$

Local scalar products $\sigma_q := (\mathbf{t}^{(s,q)})^T \underline{z}^{(s,q)}$ can be computed in parallel

Total sum

$$\sigma = \sigma_1 + \cdots + \sigma_p$$

requires global communication (MPI_Reduce/MPI_Bcast respectively OpenMP reduce+).

Parallel Vector Update

vector updates can be performed in parallel without any communication:

- if $\underline{r}$ and $\underline{z}$ are both inconsistent, then

$$\underline{r} := \underline{r} - \alpha \underline{z} \Longleftrightarrow \underline{r}^{(s,q)} := \underline{r}^{(s,q)} - \alpha \underline{z}^{(s,q)} \text{ for all } q = 1, \ldots, p$$

- if **r** and **z** are both **consistent**, then

$$\mathbf{r} := \mathbf{r} - \alpha \mathbf{z} \Longleftrightarrow \mathbf{r}^{(s,q)} := \mathbf{r}^{(s,q)} - \alpha \mathbf{z}^{(s,q)} \text{ for all } q = 1, \ldots, p$$

The Parallel CG Method

- use parallel matrix splitting for $\underline{A}$
- suppose that the initial guess for $\mathbf{x}$ is **consistent**
- suppose that the right hand side $\underline{b}$ is inconsistent

$\mathbf{x}$ initial guess
$\underline{r} = \underline{b} - \underline{A}\mathbf{x}$

The Parallel CG Method

- use parallel matrix splitting for $\underline{A}$
- suppose that the initial guess for $\mathbf{x}$ is **consistent**
- suppose that the right hand side $\underline{b}$ is inconsistent

$\mathbf{x}$ initial guess
$\underline{r} = \underline{b} - \underline{A}\mathbf{x}$
$\mathbf{t} = \underline{r}$ (communication to make $\underline{r}$ consistent)

The Parallel CG Method

- use parallel matrix splitting for $\underline{A}$
- suppose that the initial guess for $\mathbf{x}$ is **consistent**
- suppose that the right hand side $\underline{b}$ is inconsistent

$\mathbf{x}$ initial guess
$\underline{r} = \underline{b} - \underline{A}\mathbf{x}$
$\mathbf{t} = \underline{r}$ (communication to make $\underline{r}$ consistent)
$\rho = \mathbf{t}^T \underline{r}$ (global communication)

The Parallel CG Method

- use parallel matrix splitting for $\underline{A}$
- suppose that the initial guess for $\mathbf{x}$ is **consistent**
- suppose that the right hand side $\underline{b}$ is inconsistent

$\mathbf{x}$ initial guess
$\underline{r} = \underline{b} - \underline{A}\mathbf{x}$
$\mathbf{t} = \underline{r}$ (communication to make $\underline{r}$ consistent)
$\rho = \mathbf{t}^T \underline{r}$ (global communication)
for $k = 1, 2, 3, \ldots$
    $\underline{z} = \underline{A}\mathbf{t}$

The Parallel CG Method

- use parallel matrix splitting for $\underline{A}$
- suppose that the initial guess for $\mathbf{x}$ is **consistent**
- suppose that the right hand side $\underline{b}$ is inconsistent

$\mathbf{x}$ initial guess
$\underline{r} = \underline{b} - \underline{A}\mathbf{x}$
$\mathbf{t} = \underline{r}$ (communication to make $\underline{r}$ consistent)
$\rho = \mathbf{t}^T \underline{r}$ (global communication)
for $k = 1, 2, 3, \ldots$
$\quad \underline{z} = \underline{A}\mathbf{t}$
$\quad \alpha = \rho / (\mathbf{t}^T \underline{z})$ (global communication)

The Parallel CG Method

- use parallel matrix splitting for $\underline{A}$
- suppose that the initial guess for $\mathbf{x}$ is **consistent**
- suppose that the right hand side $\underline{b}$ is inconsistent

$\mathbf{x}$ initial guess
$\underline{r} = \underline{b} - \underline{A}\mathbf{x}$
$\mathbf{t} = \underline{r}$ (communication to make $\underline{r}$ consistent)
$\rho = \mathbf{t}^T \underline{r}$ (global communication)
for $k = 1, 2, 3, \ldots$
$\quad \underline{z} = \underline{A}\mathbf{t}$
$\quad \alpha = \rho / (\mathbf{t}^T \underline{z})$ (global communication)
$\quad \mathbf{x} = \mathbf{x} + \alpha \mathbf{t}$
$\quad \underline{r} = \underline{r} - \alpha \underline{z}$
$\quad \rho_{old} = \rho$

The Parallel CG Method

- use parallel matrix splitting for $\underline{A}$
- suppose that the initial guess for $\mathbf{x}$ is **consistent**
- suppose that the right hand side $\underline{b}$ is inconsistent

$\mathbf{x}$ initial guess
$\underline{r} = \underline{b} - \underline{A}\mathbf{x}$
$\mathbf{t} = \underline{r}$ (communication to make $\underline{r}$ consistent)
$\rho = \mathbf{t}^T \underline{r}$ (global communication)
for $k = 1, 2, 3, \ldots$
$\quad \underline{z} = \underline{A}\mathbf{t}$
$\quad \alpha = \rho / (\mathbf{t}^T \underline{z})$ (global communication)
$\quad \mathbf{x} = \mathbf{x} + \alpha \mathbf{t}$
$\quad \underline{r} = \underline{r} - \alpha \underline{z}$
$\quad \rho_{old} = \rho$
$\quad \mathbf{t} = \underline{r}$ (communication to make $\underline{r}$ consistent)

The Parallel CG Method

- use parallel matrix splitting for $\underline{A}$
- suppose that the initial guess for $\mathbf{x}$ is **consistent**
- suppose that the right hand side $\underline{b}$ is inconsistent

$\mathbf{x}$ initial guess
$\underline{r} = \underline{b} - \underline{A}\mathbf{x}$
$\mathbf{t} = \underline{r}$ (communication to make $\underline{r}$ consistent)
$\rho = \mathbf{t}^T \underline{r}$ (global communication)
for $k = 1, 2, 3, \ldots$
$\quad \underline{z} = \underline{A}\mathbf{t}$
$\quad \alpha = \rho / (\mathbf{t}^T \underline{z})$ (global communication)
$\quad \mathbf{x} = \mathbf{x} + \alpha \mathbf{t}$
$\quad \underline{r} = \underline{r} - \alpha \underline{z}$
$\quad \rho_{old} = \rho$
$\quad \mathbf{t} = \underline{r}$ (communication to make $\underline{r}$ consistent)
$\quad \rho = \mathbf{t}^T \underline{r}$ (global communication)

The Parallel CG Method

- use parallel matrix splitting for $\underline{A}$
- suppose that the initial guess for $\mathbf{x}$ is **consistent**
- suppose that the right hand side $\underline{b}$ is inconsistent

$\mathbf{x}$ initial guess
$\underline{r} = \underline{b} - \underline{A}\mathbf{x}$
$\mathbf{t} = \underline{r}$ (communication to make $\underline{r}$ consistent)
$\rho = \mathbf{t}^T \underline{r}$ (global communication)
for $k = 1, 2, 3, \ldots$
$\quad \underline{z} = \underline{A}\mathbf{t}$
$\quad \alpha = \rho / (\mathbf{t}^T \underline{z})$ (global communication)
$\quad \mathbf{x} = \mathbf{x} + \alpha \mathbf{t}$
$\quad \underline{r} = \underline{r} - \alpha \underline{z}$
$\quad \rho_{old} = \rho$
$\quad \mathbf{t} = \underline{r}$ (communication to make $\underline{r}$ consistent)
$\quad \rho = \mathbf{t}^T \underline{r}$ (global communication)
$\quad \beta = \rho / \rho_{old}$
$\quad \mathbf{t} = \mathbf{t} + \beta \mathbf{t}$
end

Properties of the CG Method

1. the number of iteration steps for the CG can be measured by the condition number

$$\kappa_A = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}$$

2. if $\|v\|_A := \sqrt{v^T A v}$, then after l steps the approximate solution x_l satisfies

$$\|x - x_l\|_A \leqslant 2 \left(\frac{\sqrt{\kappa_A} - 1}{\sqrt{\kappa_A} + 1} \right)^l \|x - x_0\|_A$$

3. Often κ_A is large. One usually computes a *preconditioner* $P = GG^T \approx A$ and replaces A by the preconditioned system $\hat{A} = G^{-1} A G^{-T}$.

The Parallel Preconditioned CG Method

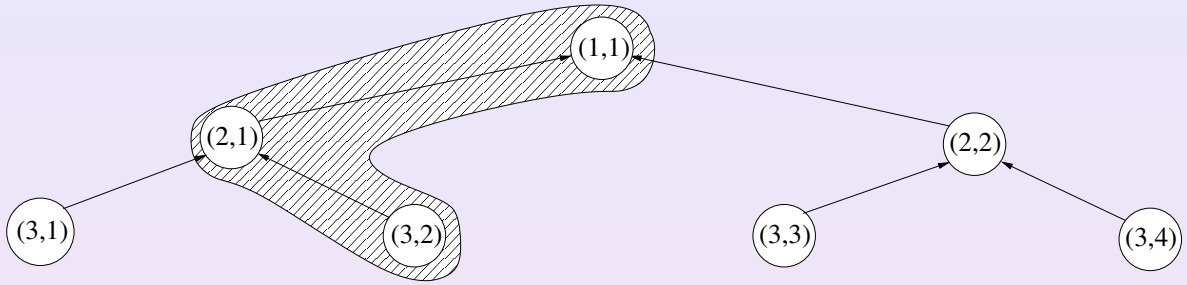
$\mathbf{x}$ initial guess
$\underline{r} = \underline{b} - \underline{A}\mathbf{x}$
$\mathbf{t} = \underline{r}$ (communication to make $\underline{r}$ consistent)
Solve $P\mathbf{u} = \mathbf{t}$ (may require communication)
$\rho = \mathbf{u}^T \underline{r}$ (global communication)
for $k = 1, 2, 3, \ldots$
    $\underline{z} = \underline{A}\mathbf{t}$
    $\alpha = \rho / (\mathbf{t}^T \underline{z})$ (global communication)
    $\mathbf{x} = \mathbf{x} + \alpha\mathbf{t}$
    $\underline{r} = \underline{r} - \alpha\underline{z}$
    $\rho_{old} = \rho$
    $\mathbf{t} = \underline{r}$ (communication to make $\underline{r}$ consistent)
    Solve $P\mathbf{u} = \mathbf{t}$ (may require communication)
    $\rho = \mathbf{u}^T \underline{r}$ (global communication)
    $\beta = \rho / \rho_{old}$
    $\mathbf{t} = \mathbf{u} + \beta\mathbf{t}$
end

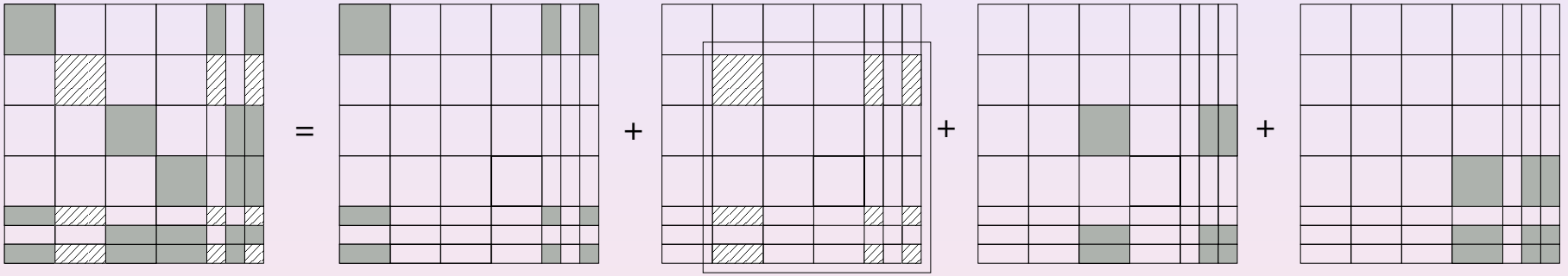
Technische Universität Braunschweig

Outline

Parallel Preconditioner

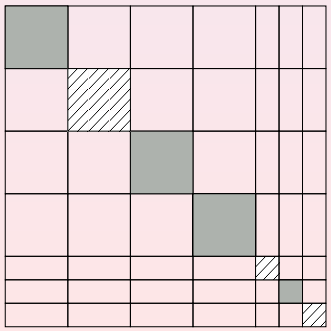
Parallel Matrix Splitting





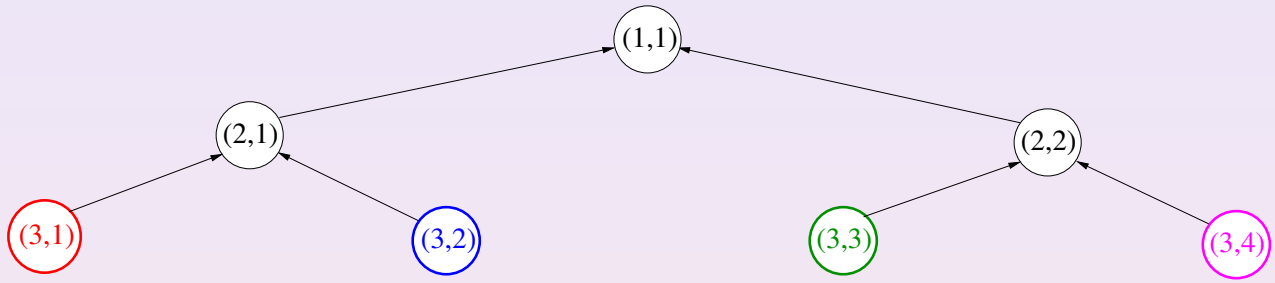
$\underline{A}$ $\underline{A}^{(3,1)}$ $\underline{A}^{(3,2)}$ $\underline{A}^{(3,3)}$ $\underline{A}^{(3,4)}$

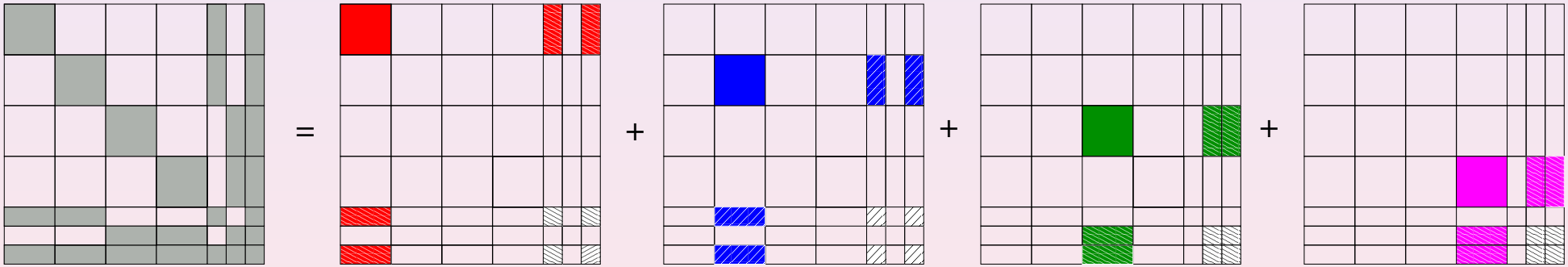
Preconditioner of the following block structure will not involve communication



Parallel Incomplete LU Factorization

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \rightarrow \begin{bmatrix} L_{11} \\ L_{21} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \end{bmatrix}$$

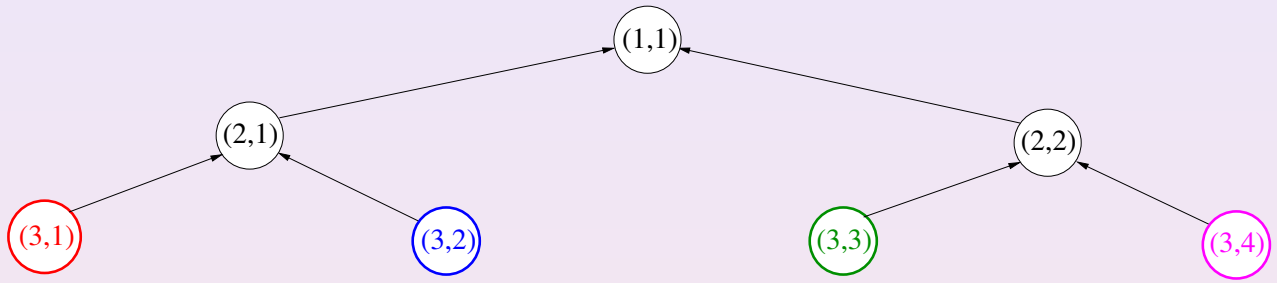


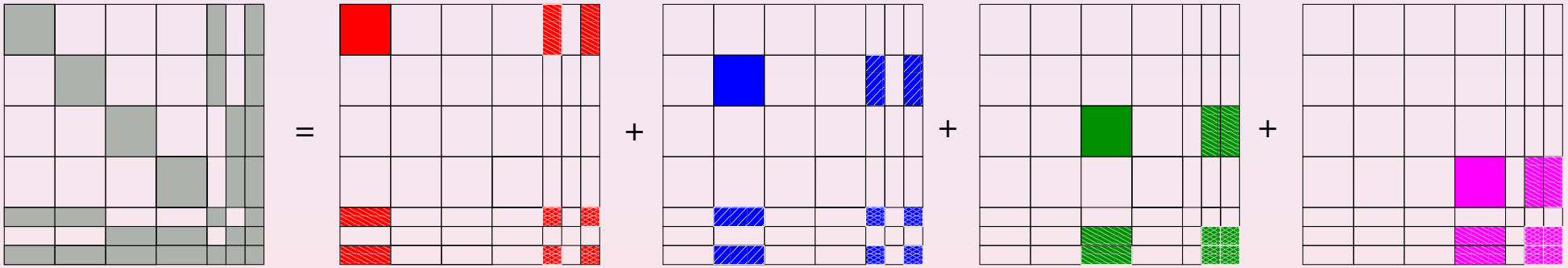


1. Initially, compute for $\underline{A}^{(s,1)}, \ldots, \underline{A}^{(s,p)}$ an approximate LU decomposition (say, dropping entries of small size), stop ILU when the leaf part is completed

Parallel Incomplete LU Factorization

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \approx \begin{bmatrix} L_{11} & 0 \\ L_{21} & I \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & S_{22} \end{bmatrix}, \text{ where } S_{22} = A_{22} - L_{21} U_{12}$$

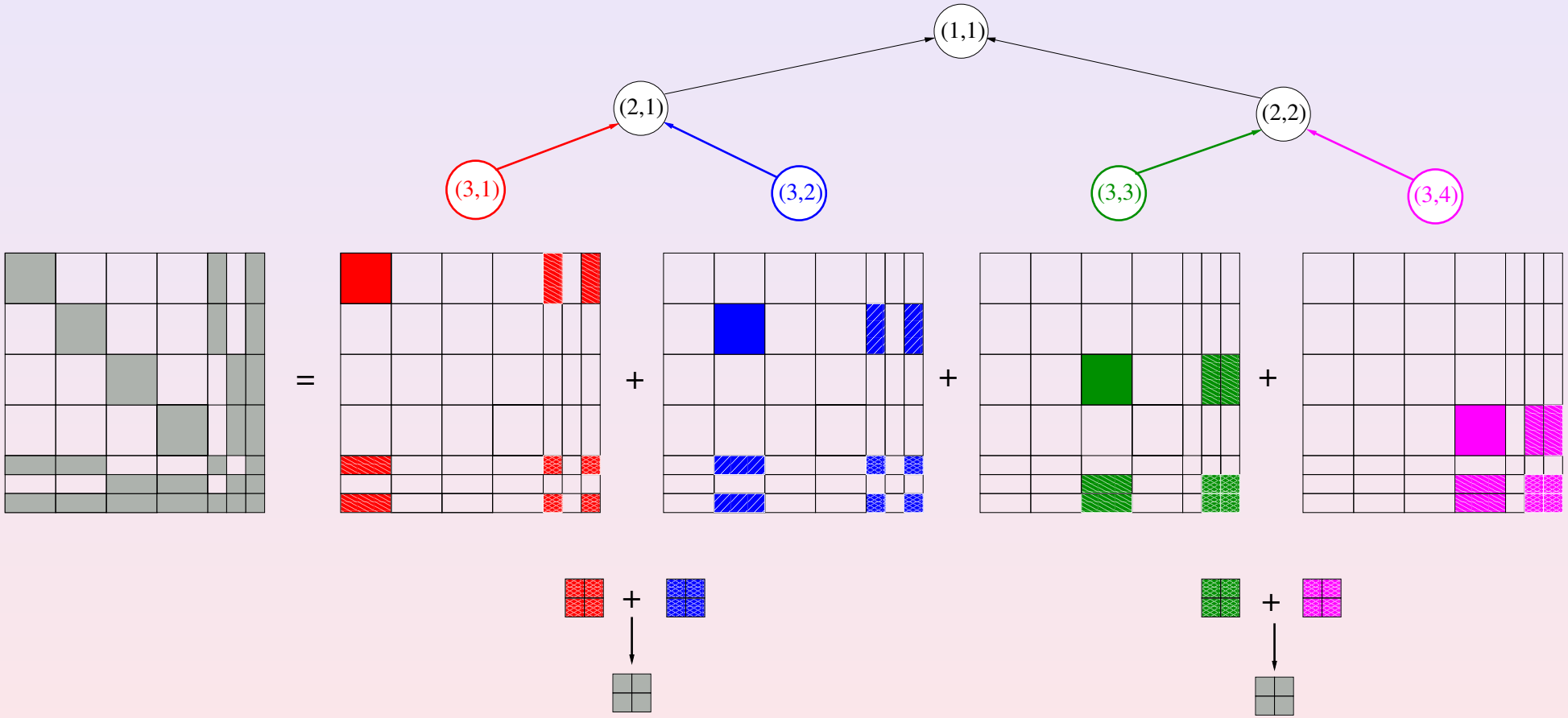




2. Compute local approximate Schur complement from A, L and U

Parallel Incomplete LU Factorization

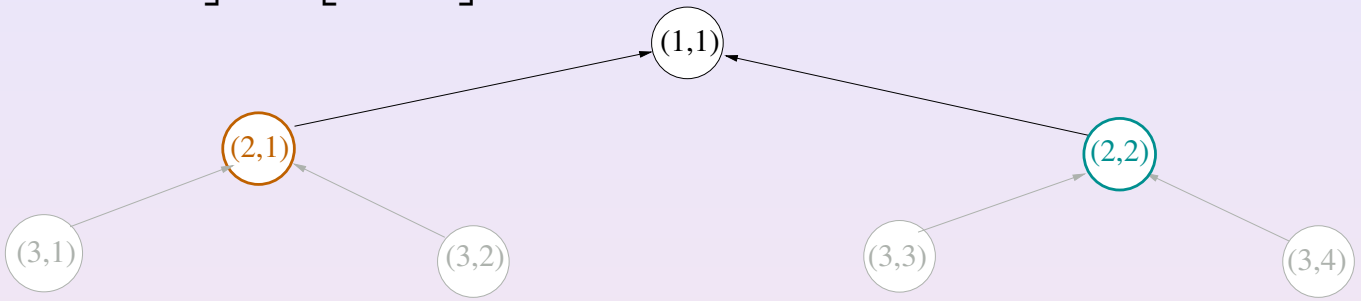
$$\underline{A}^{(s-1,1)} = \underline{A}^{(s,1)} + \underline{A}^{(s,2)}, \ldots, \underline{A}^{(s-1,p/2)} = \underline{A}^{(s,p-1)} + \underline{A}^{(s,p)}$$

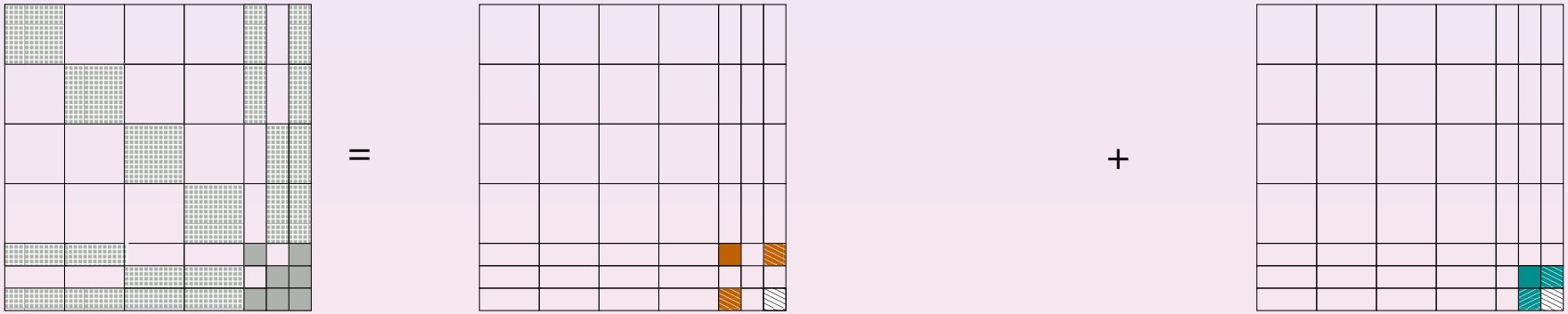


3. Exchange data of local Schur complements and merge them

Parallel Incomplete LU Factorization

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \rightarrow \begin{bmatrix} L_{11} \\ L_{21} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \end{bmatrix}$$

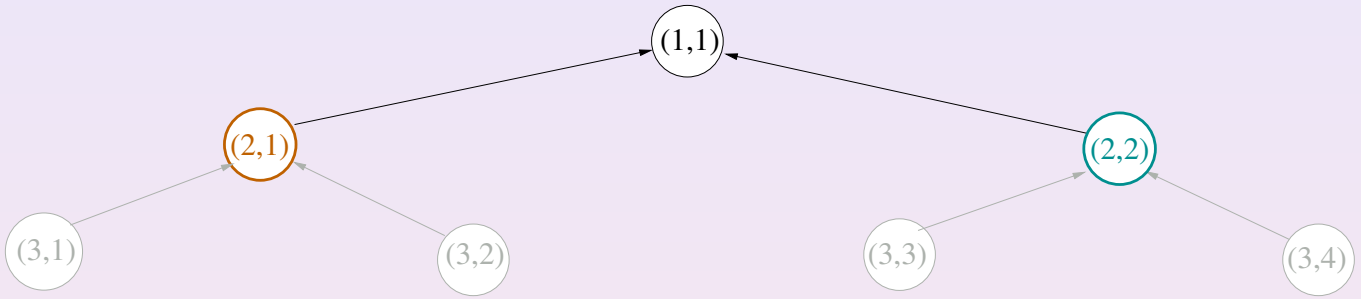


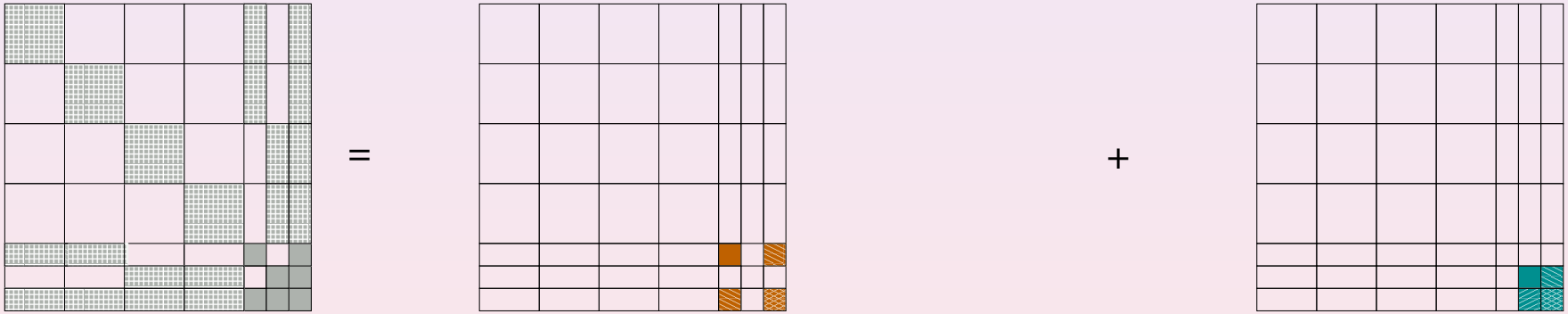


4. Repeat the same algorithms with $p/2$ processors (threads) on the next higher level,i.e., compute ILU until the leading block is finished

Parallel Incomplete LU Factorization

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \approx \begin{bmatrix} L_{11} & 0 \\ L_{21} & I \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & S_{22} \end{bmatrix}, \text{ where } S_{22} = A_{22} - L_{21} U_{12}$$

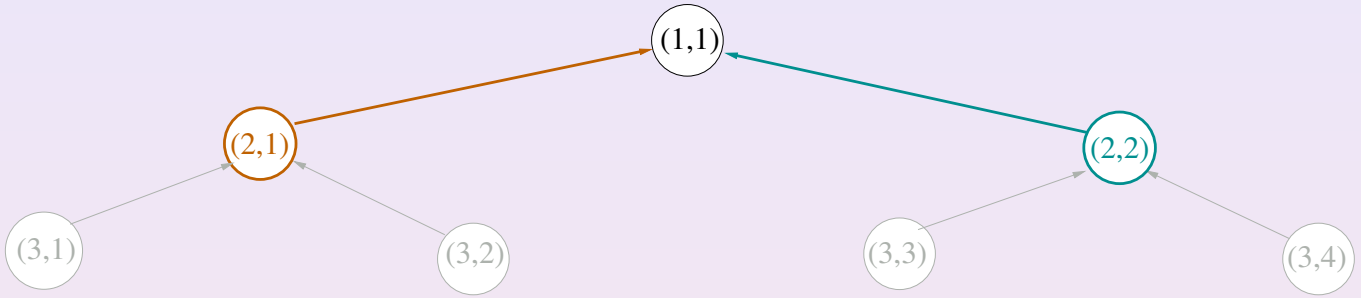


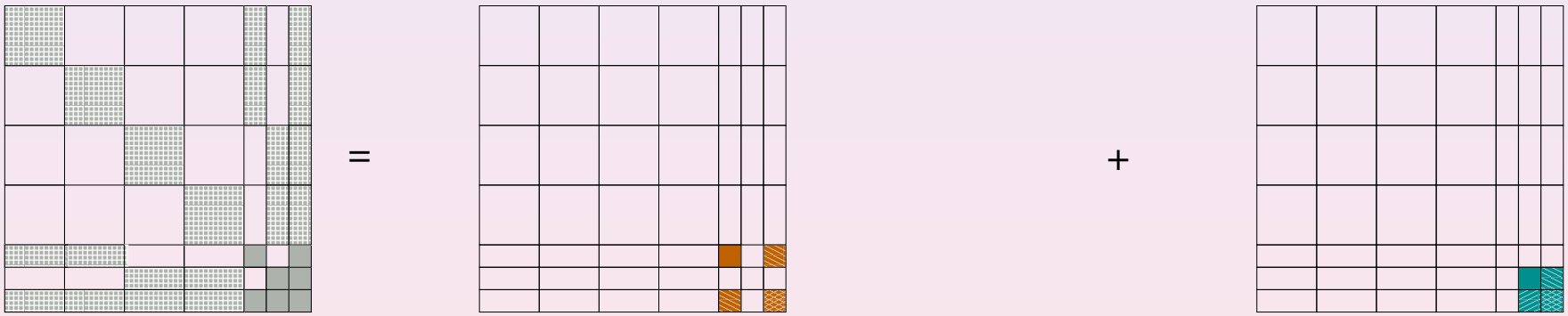


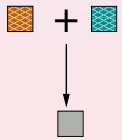
5. Again compute local approximate Schur complement

Parallel Incomplete LU Factorization

$$\underline{A}^{(s-2,1)} = \underline{A}^{(s-1,1)} + \underline{A}^{(s-1,2)}, \ldots, \underline{A}^{(s-2,p/4)} = \underline{A}^{(s-1,p/2-1)} + \underline{A}^{(s-1,p/2)}$$



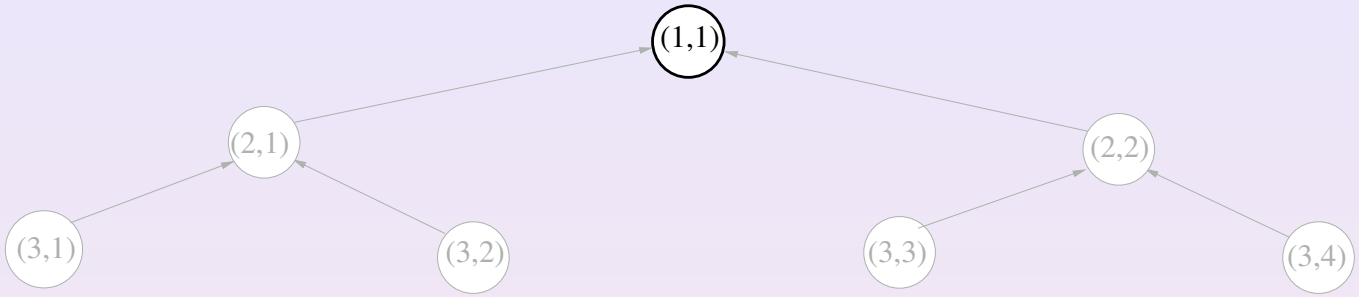




6. Again exchange data of local Schur complements and merge them

Parallel Incomplete LU Factorization

$$A \approx LU$$



7. Eventually we end up with a single system that is computed with one processor (thread) only

Parallel Incomplete LU Factorization

- The ILU is computed level by level, each time only using half as many processors (or threads) as before
- the exchange of the local Schur complements requires that two processors (resp. threads) exchange their data.
- If we use OpenMP (shared memory), the local L and U factors are available to **all** threads
- if we use MPI (distributed memory), then merging the data leads to only **one out of two** processors to keep the data and the factorization.
- We need $s = \log_2 p$ local data exchanges

Parallel Forward / Backward Substitution

- forward substitution $L\mathbf{v} = \mathbf{t}$ works analogously to the ILU
  1. each processors or thread can compute a local forward substitution on the leaf part.
  2. two processors (threads) with common ancestor in the task tree merge their data for the remaining part
  3. on the next level the right hand side is updated and the forward substitution is repeated
  4. this procedure is repeated until the root node is reached.

- backward substitution $U\mathbf{u} = \mathbf{v}$ works in reverse manner
  1. the backward substitution starts at the root node and the result is provided to its children
  2. Two threads (resp. processors) collect the root node data, update their individual right hand sides and their own backward substitution
  3. this procedure is continued until the leaf level is reached

- each single forward and each single backward substitution steps requires as many data exchanges as the ILU, except that less data is exchanged

Outline

What did we learn today?

- discretized PDEs lead to large sparse linear systems
- partitioning the graph using nested dissection leads to a lay out suitable for parallel computations
- the hierarchy obtained from nested dissection can be described by a binary tree
- nested dissection partitioning leads to a hierarchy of data partitioning
  - parallel matrix splitting
  - consistent and inconsistent vectors
- data structures can be used for parallel iterative solution of linear systems
  - matrix-vector multiplication
  - scalar product, vector updates
  - incomplete LU factorization