Parallel Numerics

Prof. Dr. Thomas Huckle *

July 2, 2006

*Technische Universität München, Institut für Informatik

Contents

1 Introduction **4**
1.1 Computer Science Aspects of Parallel Numerics 4
1.1.1 Parallelism in CPU 4
1.1.2 Memory Organization 5
1.1.3 Parallel Processors 6
1.1.4 Performance Analysis 8
1.1.5 Further Keywords: 9
1.2 Numerical Problems 10
1.3 Data Dependency Graphs 11
1.3.1 Directed Graph G=(E,V) 11
1.3.2 Dependency Graphs of Iterative Algorithms 13
1.3.3 Dependency graph for solving a triangular linear system 17

2 Elementary Linear Algebra Problems **19**
2.1 BLAS – Basic Linear Algebra Subroutines Program package . 19
2.2 Analysis of Matrix-Vector product 22
2.2.1 Vectorization 23
2.2.2 Parallelization by building blocks 23
2.2.3 $c = Ab$ for banded matrix 25
2.3 Analysis of the Matrix-Matrix-product 27

3 Linear Equations with dense matrices **29**
3.1 Gaussian Elimination: Basic facts 29
3.2 Vectorization of the Gaussian Elimination 32
3.3 Gaussian Elimination in Parallel 34
3.3.1 Crout method 35
3.3.2 left looking GE: 35
3.3.3 Right looking / Gaussian Elimination standard 36
3.4 QR-Decomposition with Householder matrices 36
3.4.1 QR-decomposition 36
3.4.2 Householder method for QR 37
3.4.3 Householder method in parallel 38

4 Linear Equations with sparse matrices **39**
4.1 General properties of sparse matrices 39
4.1.1 Storage in coordinate form 39
4.1.2 Compressed Sparse Row Format: CSR 40
4.1.3 Improving CSR 40
4.1.4 Diagonalwise storage 41

4.1.5 rectangular, rowwise storage scheme . . . 41
4.1.6 Jagged diagonal form . . . 42
4.2 Sparse Matrices and Graphs . . . 43
4.2.1 $A = A^T > 0$ (n $\times$ n - matrix) symmetric . . . 43
4.2.2 A non symmetric: directed graph . . . 44
4.2.3 Dissection form preserved during GE . . . 45
4.3 Reordering . . . 47
4.3.1 Smaller Bandwidth by Cuthill Mckee-Algorithm . . . 47
4.3.2 Dissection Reordering . . . 49
4.3.3 Algebraic pivoting: during GE . . . 50
4.4 Gaussian Elimination in Graph . . . 53
4.5 Different direct solvers . . . 55

5 Iterative methods for sparse matrices 57
5.1 stationary methods . . . 57
5.1.1 Richardson Iteration . . . 57
5.1.2 Better splitting of A . . . 58
5.1.3 Jacobi (Diagonal) - Splitting: . . . 58
5.1.4 Gauss-Seidel method by improving convergence . . . 59
5.2 Nonstationary Methods . . . 60
5.2.1 Let A symmetric positive definite $A = A^T > 0$ (spd.) . 60
5.2.2 Improving the gradient method $\rightarrow$ conjugate gradients 63
5.2.3 GMRES for General Matrix A, not spd . . . 65
5.2.4 Convergence of cg. or GMRES . . . 67

6 Collection remaining problems 70
6.1 Domain Decomposition Methods for Solving PDE . . . 70
6.2 Parallel Computation of the Discrete Fourier Transformation . 71
6.3 Parallel Computation of Eigenvalues . . . 73

Literature:
Numerical Linear Algebra for High Performance Computers Dongarra,...

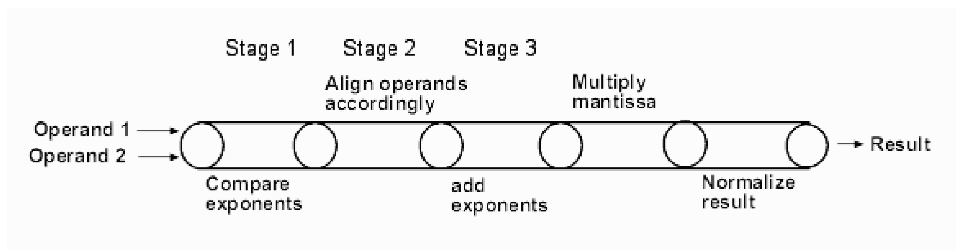
1 Introduction

1.1 Computer Science Aspects of Parallel Numerics

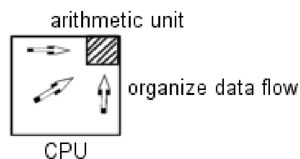
1.1.1 Parallelism in CPU

Elementary operations in CPU are carried out in pipelines:
Divide a task into a sequence of smaller tasks.
Each small task is executed on a piece of hardware, that operates concurrently with the other stages of the pipeline.

Example: Multiplication

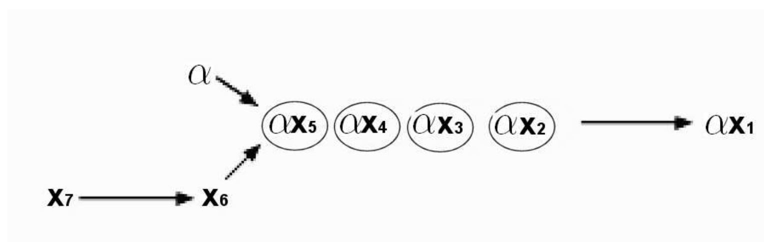


Advantage: If pipeline is filled, per clock one result comes out. All multiplications should be organized such that the pipeline is always filled!

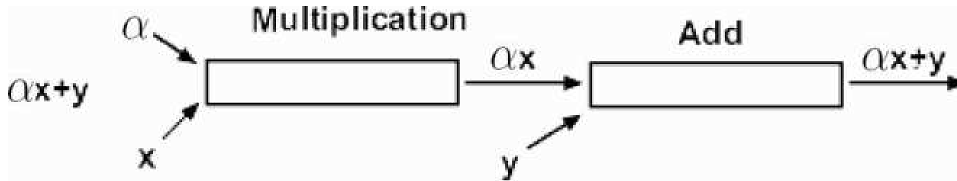


If pipeline is empty, it is not efficient.

Special Case: Vector instruction: for set of data the same operation has to be executed: $\alpha \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$ Cost: Startup time + vector length*clock period

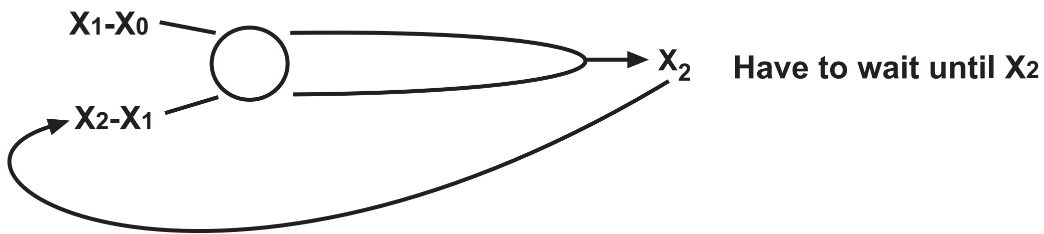


Chaining: Combine pipelines directly:



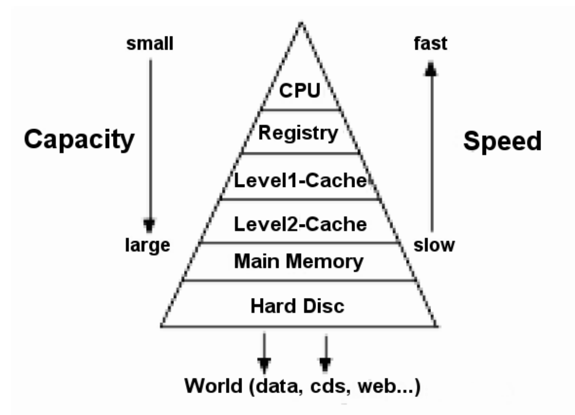
advantage: $\textit{total cost} = \underbrace{\textit{startup time}}_{\textit{longer}} + \textit{vector length} * \textit{clock period}$

Problem: Data Dependency
Fibonacci: $x_0 = 0,\ x_1 = 1,\ x_2 = x_1 + x_0, \ldots,\ x_i = x_{i-1} + x_{i-2}$



Next pair has to wait until x_2 / pipeline is empty in each step

1.1.2 Memory Organization



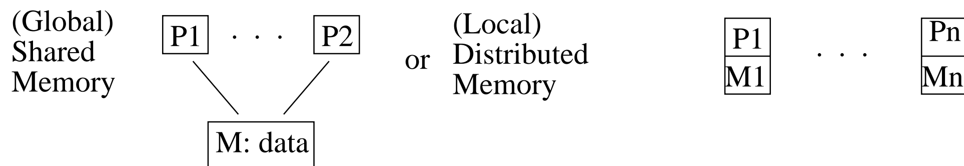
Cache idea: Buffer between large slow memory and small fast memory.
By considering the flow of the last used data, we try to predict which data will be requested in the next step:

- keep the last used data in cache for fast access
- keep also the neighbourhood of this data in cache

Cache hit: CPU looks for data and data is in cache and finds it
Cache miss: Data is not in cache: look in main memory and copy new page in the cache

1.1.3 Parallel Processors

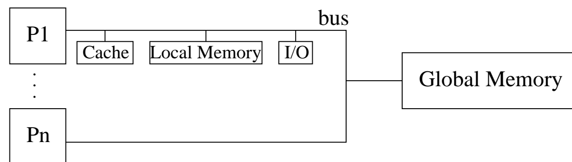
MIMD-Architecture: Multiple Instruction-Multiple Data
(global)shared memory (P=processors, M=memory):



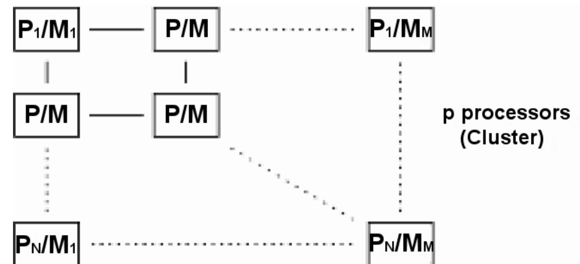
or virtual shared memory:
physically distributed data but organized as shared memory

Topology of processors/memory: Interconnection (shared memory)

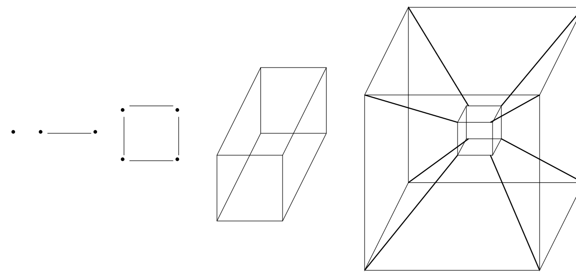
Bus:



Mesh: (distributed memory)



Hypercube:



data dependency: log(N)
Shared memory communication for different processors by:

synchronization: e.g barrier $\begin{array}{c|} \textit{halt} \\ p_1 \\ \vdots \\ p_n \end{array}$ $\longrightarrow$ continue iff all completed

MPI: Message Passing Interface: *Communication library*
for C, C++, FORTRAN

Compiling: mpicc <options> prog.c

Start: mpirun - arch <architecture> - up <up> prog

Commands: MPI_Send
MPI_Bcast
MPI_Recv
MPI_Gather
MPI_Barrier

1.1.4 Performance Analysis

computation speed:

$r = \frac{N}{t}$ Mflops $\quad$ N floating point operations in t microseconds

or by known speed: $\quad \rightarrow \quad t = \frac{N}{r}$flops

Amdahl's law: Algorithm takes N flops

fraction f carried out with speed of V Mflops (good parallel)
fraction $1-f$ carried out with speed of S Mflops (bad parallel)
fraction f is well-suited for parallel execution ; $1-f$ is not.

Total CPU-time:

$$t = \frac{f \cdot N}{V} + \frac{(1-f) \cdot N}{S} = N \left(\frac{f}{V} + \frac{1-f}{S} \right) \quad \text{microseconds}$$

Overall Speed (Performance):

$$r = \frac{N}{t} = \frac{1}{\frac{f}{V} + \frac{1-f}{S}} \text{ Mflops} \qquad \underline{\text{Amdahl's law}}$$

Interpretation: f must be close to 1 in order to benefit significantly from parallelism

Speedup by using p parallel processors for a given job:
$t_j :=$ wall clock time to execute the job on j parallel processor
Speedup: $S_p = t_1/t_p \qquad (ideal : t_1 = pt_p)$
Efficiency: $E_p = \frac{S_p}{p} \quad 0 \leq E_p \leq 1$
$E_p \approx 1 :$ very good parallelizable $t_p = t_1/p$ problem scales

$$t_p = \frac{\overbrace{ft_1}^{parallel}}{p} + (1-f)t_1 = t_1 \frac{f + (1-f)p}{p} \geq (1-f)t_1$$

$$S_p = \frac{1}{\frac{f-(1-f)p}{p}} = \frac{p}{f+(1-f)p} \qquad \underline{\text{Ware's law}}$$

$$E_p = \frac{1}{f+(1-f)p} \quad ; \quad \lim_{p\to\infty} E_p = \lim_{p\to\infty} \frac{1}{f+(1-f)p} \to 0$$

Assume that the given problem can be solved in 1 unit of time on a parallel machine with p processors.

A uniprocessor would perform $(1-f) + f \cdot p$
Speedup:

$$S_{pf} = \frac{t_1}{t_p} = \frac{1-f+fp}{1} = p + (1-p)(1-f) \qquad \underline{\text{Gustafson's law}}$$

$$E_{pf} = \frac{S_p}{p} = \frac{1-f}{p} + f \stackrel{p\to\infty}{\longrightarrow} f \ (\textit{only theoretical use!})$$

1.1.5 Further Keywords:

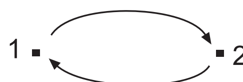
- An algorithm is scaling iff with p processors we can reduce the operation time by a factor of p.
  Larger problem can be solved in the same time by using more processors
  (This means: speedup $\approx p$, efficiency ≈ 1)

- load balancing: The job has to be distributed on different processors such that all processors are busy: Avoid idle processors

- deadlock: two or more processors are waiting indefinitely for an event that can be caused only by one of the waiting processors

- data dependency: compute

$$C = A + B \qquad (1)$$
$$Z = C \cdot X + Y \qquad (2)$$



Each waiting fot the results of the other one

(2) can be computed only <u>after</u> (1)
example loop:

$$for(i=1; \quad i \leq n; \quad i{+}{+}) \qquad a[i] = b[i] + a[i-1] + c[i]$$

(strongly sequential)

1.2 Numerical Problems

vectors $x,\, y \in \mathbb{R}^n$

dot product (inner product) $x^T y = (x_1, \ldots, x_n) \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \sum_{i=1}^{n} x_i y_i$

sum of vectors: $x + \alpha y = \begin{pmatrix} x_1 + \alpha y_1 \\ \vdots \\ x_n + \alpha y_n \end{pmatrix}$

outer product: $xy^T = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} (y_1, \ldots, y_m) = \begin{pmatrix} x_1 y_1 & \ldots & x_1 y_m \\ \vdots & & \vdots \\ x_n y_1 & \ldots & x_n y_m \end{pmatrix}$

matrix product: $\mathrm{A} \in \mathbb{R}^{n,k}$, $\mathrm{B} \in \mathbb{R}^{k,m}$, $\mathrm{C} = \mathrm{A} \cdot \mathrm{B} \in \mathbb{R}^{n,m}$

$$\begin{pmatrix} a_{11} & \ldots & a_{1k} \\ \vdots & & \vdots \\ a_{n1} & \ldots & a_{nk} \end{pmatrix} \begin{pmatrix} b_{11} & \ldots & b_{1m} \\ \vdots & & \\ b_{k1} & & b_{km} \end{pmatrix} = \begin{pmatrix} c_{11} & \ldots & c_{1m} \\ \vdots & & \\ c_{n1} & & c_{nm} \end{pmatrix}$$

with $\quad c_{ij} = \sum_{r=1}^{k} a_{ir} b_{rj}, \quad {\scriptstyle i=1,\ldots,n \atop j=1,\ldots,m}$

Solving linear equations,

e.g. triangular: $\begin{pmatrix} a_{11} & \ldots & & a_{1n} \\ 0 & a_{22} & & a_{2n} \\ \vdots & & \ddots & \\ 0 & & 0 & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$

or

$$
\begin{aligned}
a_{11}x_1 + \dots \quad + a_{1n}x_n &= b_1 \\
a_{22}x_2 + \dots + a_{2n}x_n &= b_2 \\
\ddots \qquad\qquad &\quad \vdots \\
a_{nn}x_n &= b_n
\end{aligned}
$$

solution : $x_n = \frac{b_n}{a_{nn}}, \quad a_{n-1,n-1}x_{n-1} + a_{n-1,n}x_n = b_{n-1}$

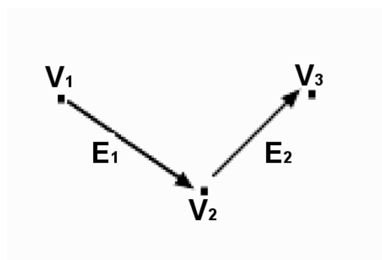
$\Rightarrow x_{n-1} = \frac{b_{n-1} - a_{n-1,n}x_n}{a_{n-1,n-1}}$

general form $x_j = \frac{b_j - \sum\limits_{k=j+1}^{n} a_{jk}x_k}{a_{jj}} \qquad for \quad j = n, ..., 1$

- Gaussian Elimination, LU-Decomposition (Cholesky-Decomposition)
- Least Squares problem (normal equation) $\min\limits_x \|Ax - B\|_2$
- QR -Decomposition
- Differential Equations (PDE)
- eigenvalues, singular values, FFT

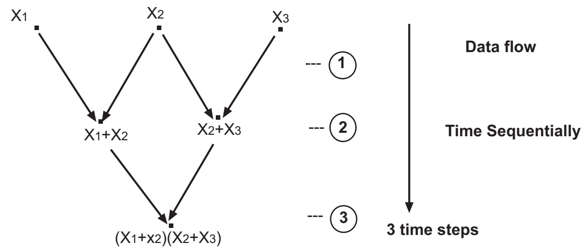
1.3 Data Dependency Graphs

1.3.1 Directed Graph G=(E,V) with edges E, vertices/nodes V



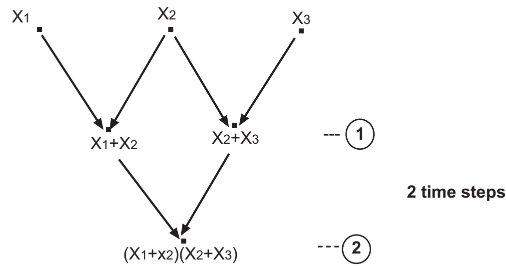
Example: Computation of $(x_1 + x_2)(x_2 + x_3) = x_1x_2 + x_2^2 + x_1x_3 + x_2x_3$

Input:

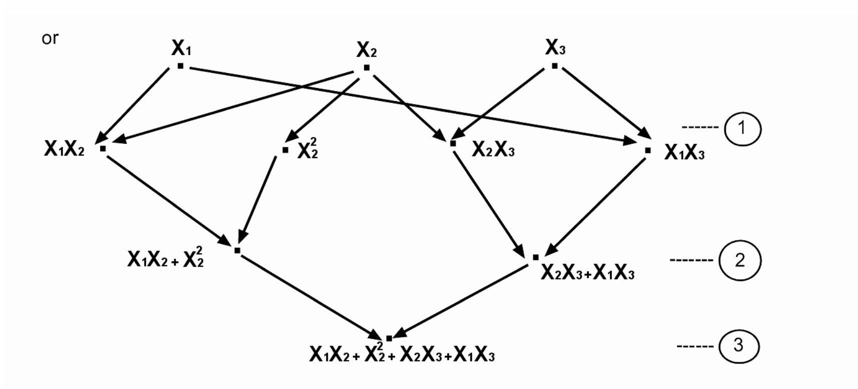


In parallel: $x_1 + x_2$ and $x_2 + x_3$ can be computed independently

Parallel:



Second equivalent formula (Parallel):



1.3.2 Dependency Graphs of Iterative Algorithms

Given: Function f, start $x^{(0)}$, $x^{(k+1)} = f(x^{(k)})$, Notation: $x^{(k+1)} \hat{=} x(k+1)$

$x^{(k)} \xrightarrow{k\to\infty} \quad \bar{x} = f(\bar{x}) \qquad$ fix point of f

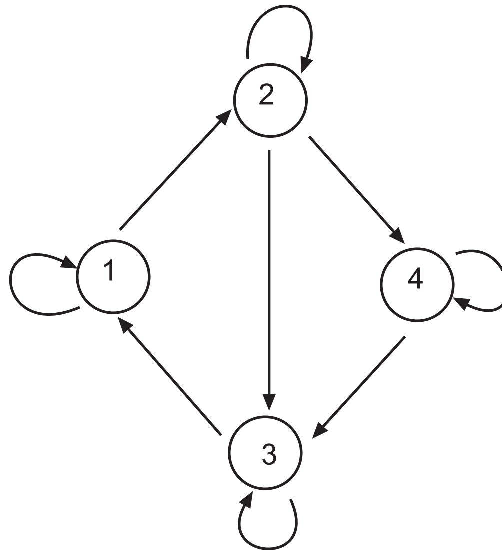
compare Newton's method: $\underline{x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)}} \to g(\bar{x}) = 0$

In vector form: $\begin{pmatrix} x_1(k+1) \\ \vdots \\ x_n(k+1) \end{pmatrix} = \vec{x}^{(k+1)} = \vec{f}\vec{x}^{(k)} = \begin{pmatrix} f_1(x_1(k)), \ldots, x_n(k) \\ \vdots \\ f_n(x_1(k)), \ldots, x_n(k) \end{pmatrix}$

Example:

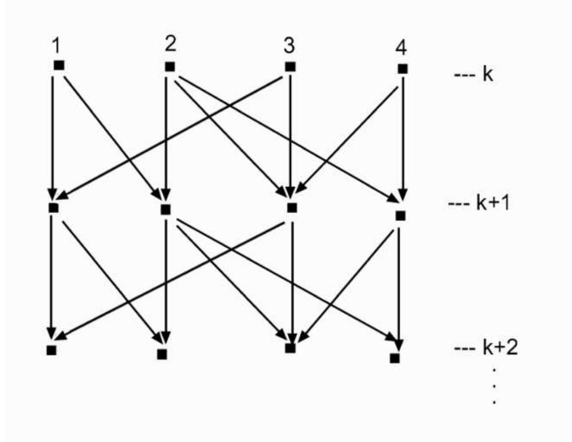
$$\begin{aligned} x_1(k+1) &= f_1(x_1(k), \quad x_3(k)) \\ x_2(k+1) &= f_2(x_1(k), \quad x_2(k)) \\ x_3(k+1) &= f_3(x_2(k), \quad x_3(k), \ x_4(k)) \\ x_4(k+1) &= f_4(x_2(k), \quad x_4(k)) \end{aligned}$$

edge i to j iff for $x_i^{(k+1)}$ we need $x_j^{(k)}$



Parallel Computation:

Dependency Graph for Iteration: Single-step or Jacobi-Iteration



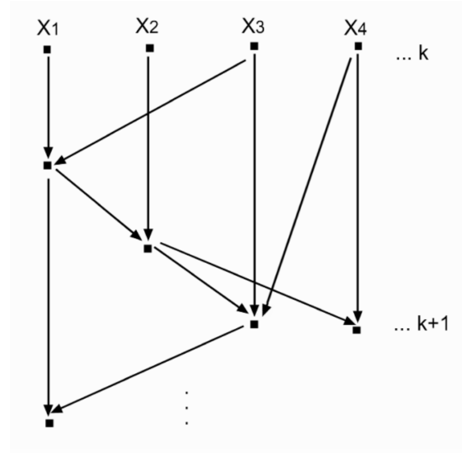
Very nice in parallel ; Convergence slow: $x^{(k)} \stackrel{k\to\infty}{\longrightarrow} \bar{x}$

Idea for accelerating the Convergence:
Use always the newest available information:

$$
\begin{aligned}
x_1(k+1) &= f_1(x_1(k),\ \ x_3(k)) \\
x_2(k+1) &= f_2(x_1(k+1),\ \ x_2(k)) \\
x_3(k+1) &= f_3(x_2(k+1),\ \ x_3(k),\ x_4(k)) \\
x_4(k+1) &= f_4(x_2(k+1),\ \ x_4(k))
\end{aligned}
$$

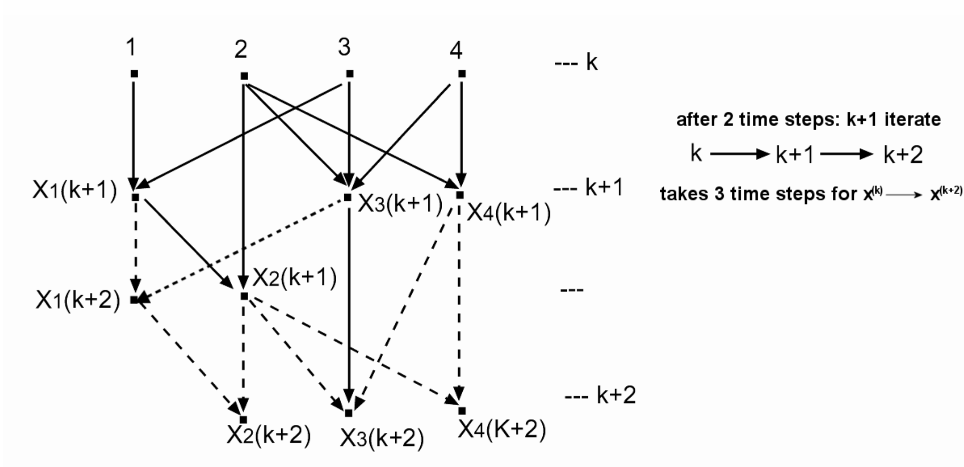
leads to much faster convergence.

Full-step or <u>Gauss-Seidel-method</u> (Drawback: loss of Parallelism:)



In this form the iteration depends on the ordering of the variables $x_1, \ldots,\ x_n$

$$
\begin{aligned}
x_1(k+1) &= f_1(x_1(k),\quad x_3(k)) \\
x_3(k+1) &= f_3(x_2(k),\quad x_3(k),\ x_4(k)) \\
x_4(k+1) &= f_4(x_2(k),\quad x_4(k)) \\
x_2(k+1) &= f_2(x_1(k+1),\quad x_2(k))
\end{aligned}
$$



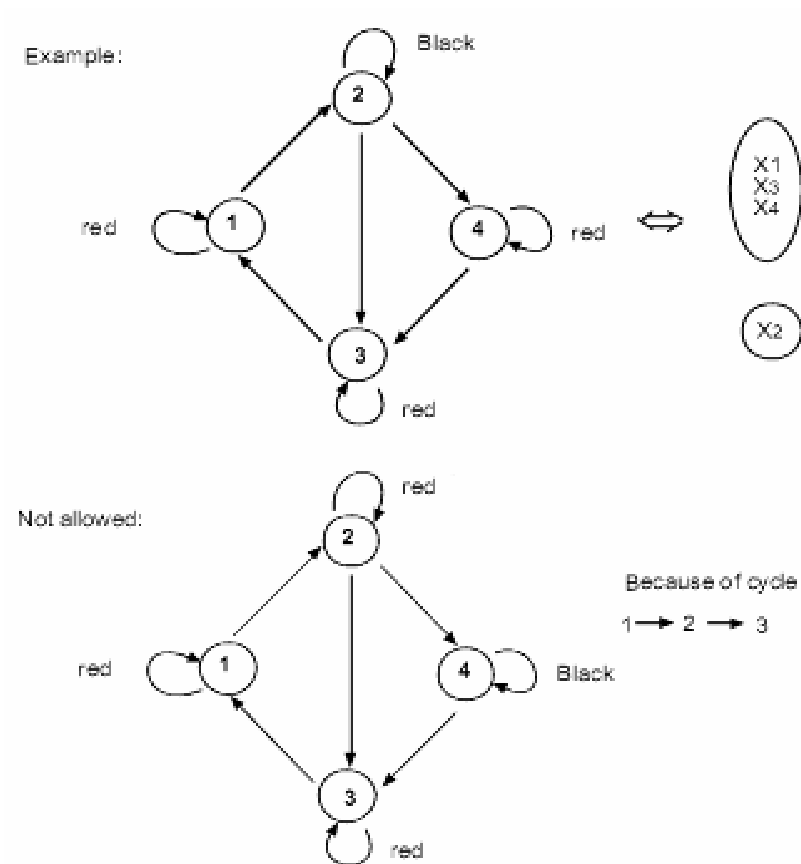
Better parallelism, but slower convergence.
→ Find "optimal" ordering with fast convergence and good in parallel.
Colouring algorithms for dependency graphs:

- use k colours for the vertices of the graph
- vertices of the same colour can be computed in parallel
- optimal colouring for minimal k, but without cycles connecting vertices of the same colour.

⇒ in subset of vertices of the same colour there are no cycles.
⇒ subgraph is a tree.

Ordering by starting with the leaves and ending with the root.

Example:



x_3 does not depend on x_1 ; x_4 does not depend on x_3
x_2 uses new computed x_1, x_3, x_4 and needs one time step

Computation uses only old information $\Rightarrow$ in parallel in one time step.

Theorem 1 *Theorem: Two statements are equivalent:*

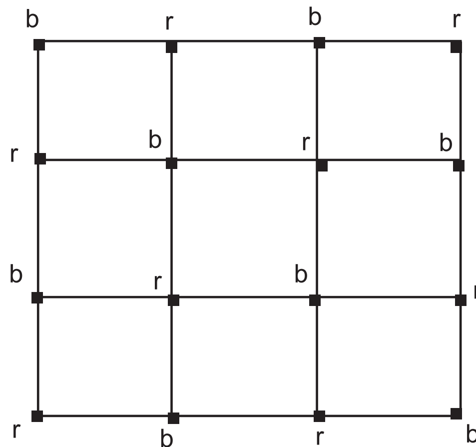
a. *There exists an ordering, such that the one Gauss-Seidel-Iteration-step takes k (time) levels*

b. *There exists a colouring with k colours, such that there is no cycle of edges of the same colour.*

Proof: Colouring in subgraph with no cycles $\rightarrow$ tree $\rightarrow$ ordering (from leaves to root) $\rightarrow$ no data dependency in subgraph $\rightarrow$ subgraph in parallel

Graph: discretization of physical problems in $\mathbb{R}$; neighbourconnections k=2

red-black-Gauss-Seidel in PDE, 2 time steps



1.3.3 Dependency graph for solving a triangular linear system

$$\begin{aligned}
a_{11}x_1 &= b_1 \\
a_{21}x_1 + a_{22}x_2 &= b_2 \\
a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \\
a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 &= b_4
\end{aligned}$$

or

$$\begin{pmatrix} a_{11} & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}$$

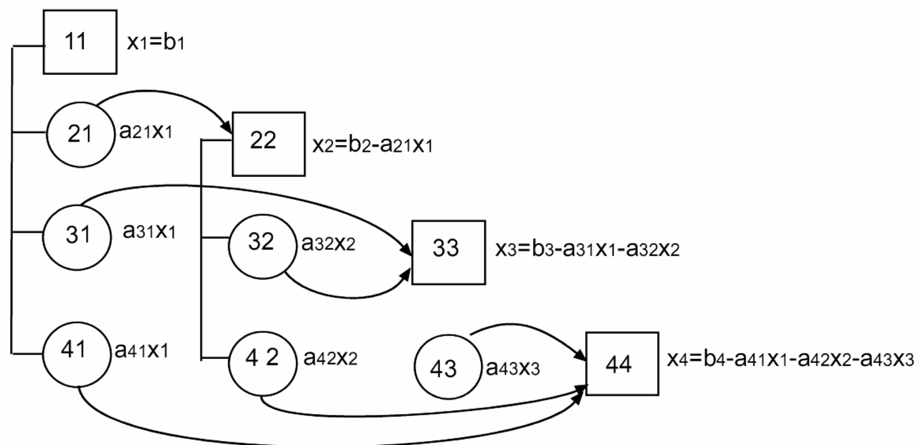
solution:

$$\begin{aligned} x_1 &= b_1/a_{11} \\ x_2 &= (b_2 - a_{21}x_1)/a_{22} \\ x_3 &= (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33} \\ x_4 &= (b_4 - a_{41}x_1 - a_{42}x_2 - a_{43}x_3)/a_{44} \end{aligned}$$

strongly sequential problem

General: $x_k = (b_k - \sum_{j=1}^{k-1} a_{kj}x_j)/a_{kk}$ for k = 1, ..., n

Dependency Graph: Assume $a_{jj} = 1$



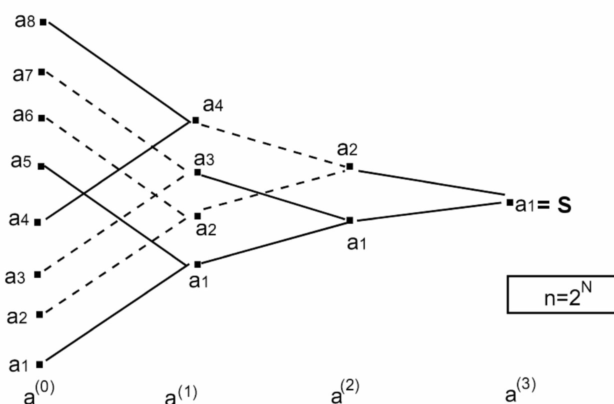
2n-1 timesteps

2 Elementary Linear Algebra Problems (dense matrices, parallel-vectorised)

2.1 BLAS – Basic Linear Algebra Subroutines Program package

Sum:

$$s = \sum_{i=1}^{n} a_i \quad \text{by } \underline{\text{fan-in process}}$$



$$a^{(k)} = \begin{pmatrix} a_1^{(k)} \\ \vdots \\ \vdots \\ a_{2^{N-k}}^{(k)} \end{pmatrix} = \begin{pmatrix} a_1^{(k-1)} \\ \vdots \\ \vdots \\ a_{2^{N-k}}^{(k-1)} \end{pmatrix} + \begin{pmatrix} a_{2^{N-k}+1}^{(k-1)} \\ \vdots \\ \vdots \\ a_{2^{N-k+1}}^{(k-1)} \end{pmatrix}$$

<u>Grouping:</u> $a_1 + \ldots + a_8 = [(a_1 + a_5) + (a_3 + a_7)] + [(a_2 + a_6) + (a_4 + a_8)]$

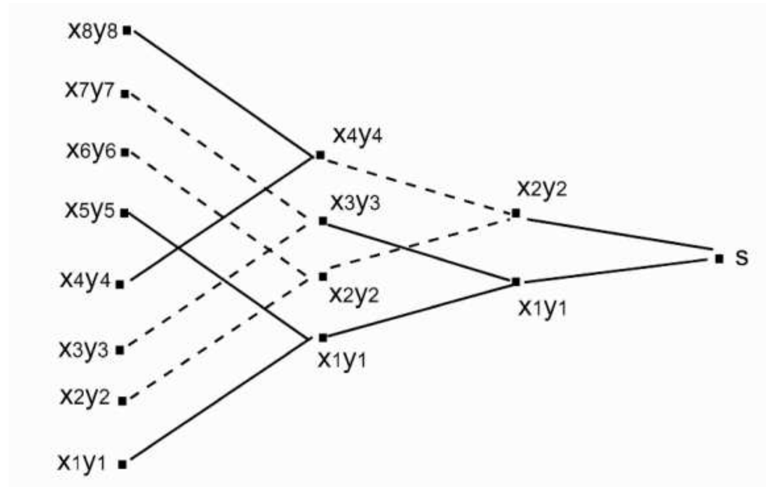
for $(k = 1;\ \underline{k \leq N};\ k++)$
    for $(j = 1;\ j \leq 2^{N-k};\ j++)$
        $a_j = a_j + a_{j+2^{N-k}};$
    end
end

full binary tree with $n = 2^N$ leaves
$\Rightarrow$ depth $\hat{=}$ time $\log n =$ N (sequential: O(n))

Level–1 BLAS: Basic Linear Algebra Subroutines with O(n) problems (Vectors only)
e.g. DOT-product by fan-in:

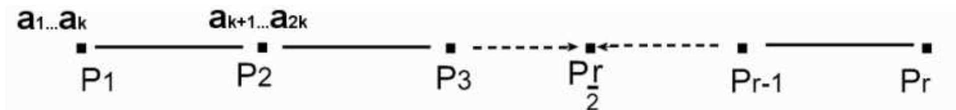
$$s = x^T y = \sum_{j=1}^{n} x_j y_j$$

Parallelization of dot-product: $\sum_{j=1}^{n} x_j y_j$



Dot-product is not very good in parallel in vectorization

<u>Other way of computing DOT-product on a special architecture:</u>
Distribute data on linear 1 dimensional processor array
with $r = n/k$ processors



Break $x_1, ..., x_n$ in r small vectors of length k.
Each processor computes $a_{j_1} b_{j_1} + \ldots + a_{j_k} b_{j_k}$

Time for this parallel computation:

k·(add + mult) i.e. time for one Addition/Multiplication

After computing this part, processor P_1/P_r sends his result to the right/left neighbour, which adds the new data to his own result, and sends the new data to his right/left neighbour until $P_{r/2}$ holds the final number.

Total time: (depending on n and r)

$$f(r) = K(add\ +\ multi) + \frac{r}{2} \cdot send + \frac{r}{2} \cdot add = \frac{n}{r} \cdot (add + mult) + r \cdot \frac{a+s}{2}$$

Minimize total time $f(r)$:

$$0 = f'_{(r)} = \ -\frac{a+m}{r^2}n + \frac{a+s}{2} \ \Rightarrow \ r = \sqrt{\frac{2(a+m)}{a+s}}.\sqrt{n} = O(\sqrt{n})$$

optimal: with $\sqrt{n}$ processors, the time is O($\sqrt{n}$)

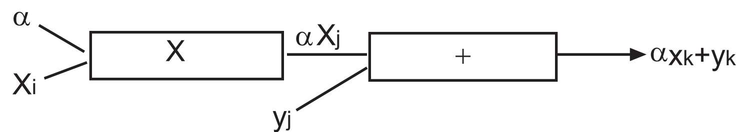
$f(\sqrt{n}) = \frac{n}{\sqrt{n}}(a+m) + \sqrt{n}\frac{a+s}{2} = O(\sqrt{n})$

Then with n processors the total time is O(log n).

Further level-1 BLAS problems

$$\begin{array}{lll} S & -\, single(precision) & \\ A\ - & \alpha & \\ X\ - & \overrightarrow{x} & :\quad \overrightarrow{y} = \alpha\overrightarrow{x} + \overrightarrow{y} \\ P\ - & + & \\ Y\ - & \overrightarrow{y} & \end{array}$$

by pipelining vectorization:



Parallelization by partitioning: $\langle 1, n\rangle = \{1, 2, 3, \dots, n\} = I_1 \cup I_2 \cup \dots \cup I_R$

$$\overrightarrow{x} = \begin{pmatrix} \underline{x_1} \\ \underline{x_2} \\ \vdots \\ x_R \end{pmatrix}, \overrightarrow{y} = \begin{pmatrix} \underline{y_1} \\ \underline{y_2} \\ \vdots \\ y_R \end{pmatrix}$$

Each processor p_j from p_1 to p_r gets $\overrightarrow{x_j}$ and $\overrightarrow{y_j}$ and computes $\alpha\overrightarrow{x_j} + \overrightarrow{y_j}$ very good vectorizable and parallelizable:

SCOPY:

$$\overrightarrow{y} = \overrightarrow{x}$$

NORM:

$$\|x\|_2 = \sqrt{\sum_{j=1}^{n} x_j^2} \qquad \text{compare DOT}$$

Level-2 BLAS: Matrix-Vector $O(n^2)$ sequentially

$$\begin{matrix} S & \text{single precision} \\ \left.\begin{matrix} G \\ E \\ M \end{matrix}\right\} & \text{general matrix : } \overrightarrow{y} = \alpha A\overrightarrow{x} + \beta\overrightarrow{y} \\ V & \text{vector} \end{matrix}$$

or solving triangular system Lx = b, L is lower triangular matrix

Level-3 BLAS: Matrix-Matrix $O(n^3)$

$$\begin{matrix} S & \text{single precision} \\ \left.\begin{matrix} G \\ E \\ M \end{matrix}\right\} & \text{general matrix : } \overrightarrow{y} = \alpha AB + \beta C \\ M & \text{matrix} \end{matrix}$$

Based on BLAS: LAPACK- subroutines for solving linear equations, least squares, QR-decomposition, eigenvalues, eigenvectors.

2.2 Analysis of Matrix-Vector product

$A = (a_{i=1..n,j=1..m}) \quad \mathbb{R}^{n,m}, \quad b \in \mathbb{R}^m,\ c \in \mathbb{R}^n,\ C = Ab$

2.2.1 Vectorization

$$\begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} a_{11} & \dots & a_{1m} \\ \vdots & & \\ a_{n1} & \dots & a_{nm} \end{pmatrix} \cdot \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} = \begin{pmatrix} a_{11}b_1 + \dots + a_{1m}b_m \\ \vdots \\ a_{n1}b_1 + \dots + a_{nm}b_m \end{pmatrix}$$

$$= \underbrace{\begin{pmatrix} \sum_{j=1}^{m} a_{1j}b_j \\ \vdots \\ \sum_{j=1}^{m} a_{nj}b_j \end{pmatrix}}_{\text{collection of DOT (rows of A)}} = \underbrace{\sum_{j=1}^{m} b_j \begin{pmatrix} a_{1j} \\ \vdots \\ a_{nj} \end{pmatrix}}_{\text{collection of SAXPY's (columns of A)} \to \text{GAXPY}}$$

(ij)– form: c = $\emptyset$;

```
for i = 1,..,n
    for j = 1,..,m
       c_i = c_i + a_ij b_j        } DOT (entries of c),  where c_i = (a_i.) · b.  i-th row of A · b
    end
end
```

(ji)–form:

```
for j = 1,..,m

 for i = 1,..,n      }  SAXPY                     }  GAXPY
    c_i = c_i + a_ij b_j }  c = c + b_j(a_.j)      }  Number of SAXPY's
 end                 }  Add j-th column of A      }  with the same vector c

end
```

Advantage of GAXPY: keep c in fast register memory
SAXPY/GAXPY good vectorizable

2.2.2 Parallelization by building blocks

Reduce Matrix-vector on smaller Matrix-vector on precessors
$\langle 1, n\rangle = \{1,2,3,..,n\} = I_1 \cup I_2 \cup I_3 ... I_R$ disjunct: $I_j \cap I_k = \emptyset$
$\langle 1, m\rangle = J_1 \cup J_2 \cup J_3 ... J_s$ $J_j \cap J_k = \emptyset$ for $j \neq k$
processor P_{rs} gets matrix $A_{rs} := A(I_r, J_s)$, $b_s = b(J_s)$, $c_r = c(I_r)$

$$I_r \begin{pmatrix} & \vert & \vert & \\ \hline & \vert & A_{rs} & \vert \\ \hline & \vert & \vert & \end{pmatrix} \cdot \begin{pmatrix} \overline{\underline{b_s}} \end{pmatrix} \} J_s = \begin{pmatrix} \overline{\underline{c_r}} \end{pmatrix} \} I_r$$

(the column block index J_s is printed above the matrix)

$$c_r = \sum_{s=1}^{S} A_{rs} \cdot b_s = \sum_{s=1}^{S} c_r^{(s)} \quad \text{for r = 1,...,R}$$

```
for r = 1,...,R
   for s = 1,...,S
      c_r^(S) = A_rs · b_s
   end
end
```

} small, independent matrix-vector products
no communication, totally parallel

```
for r = 1,...,R
   c_r = 0
   for s = 1,...,S
      c_r = c_r + c_r^(S)
   end
end
```

} blockwise collection and addition of vectors
rowwise communication (parallel)

Special case S=1:

$$c = \begin{pmatrix} \underline{A_{1\cdot}} \\ \underline{A_{2\cdot}} \\ \vdots \end{pmatrix} \cdot b = \begin{pmatrix} \underline{A_{1\cdot} \cdot b} \\ \underline{A_{2\cdot} \cdot b} \\ \vdots \end{pmatrix} \text{ no communincation between processors } \begin{pmatrix} P_1 \\ P_2 \\ \vdots \end{pmatrix} :$$

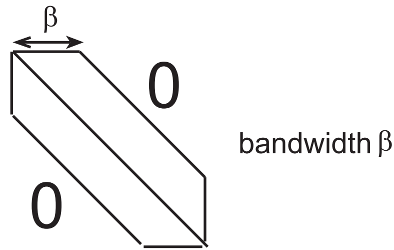
(processors independent of each other)
compute $A_{1\cdot} \cdot b$ in vectorizable form by GAXPY's

Special case R=1: $c = (A_{.1}|A_{.2}|...). \begin{pmatrix} \underline{b_1} \\ \underline{b_2} \\ \vdots \end{pmatrix} = A_{.1}b_1 + A_{.2}b_2 + ...$

$A_{.i} \cdot b_i$ independent, then collection of result $P_1 \dots P_s$ (not so good in parallel)

Rule:
1.) Vectorization - pipelining (inner most loops)
2.) Cache
3.) Parallel (outer most loops)

2.2.3 $c = Ab$ for banded matrix



e.g $\beta = 1$: tridiagonal matrix (0: main diagonal, +/-1 first upper/lower)

$$A = \begin{pmatrix} \ddots & \ddots & 0 & - & 0 \\ \ddots & \ddots & \ddots & & | \\ 0 & \ddots & \ddots & \ddots & 0 \\ | & & \ddots & \ddots & \ddots \\ 0 & - & 0 & \ddots & \ddots \end{pmatrix} \quad \text{notation} \longrightarrow$$

$$\tilde{A} = \begin{pmatrix} \tilde{a}_{10} & \tilde{a}_{11} & \tilde{a}_{1\beta} & 0 & \dots & 0 \\ \tilde{a}_{2,-1} & \tilde{a}_{20} & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & 0 \\ \vdots & \ddots & \ddots & \ddots & & \tilde{a}_{n-\beta,\beta} \\ \tilde{a}_{\beta+1,-\beta} & & \ddots & \ddots & \ddots & \\ 0 & & & \ddots & \ddots & \vdots \\ 0 & - & 0 & \tilde{a}_{n,-\beta} & \dots & \tilde{a}_{n0} \end{pmatrix}$$

$$\longrightarrow \begin{pmatrix} 0 & \dots & 0 & \tilde{a}_{1,0} & \dots & \dots & \tilde{a}_{1,\beta} \\ \vdots & / & / & \vdots & \vdots & \vdots & \vdots \\ 0 & / & & \vdots & . & . & \vdots \\ \tilde{a}_{\beta+1,-\beta} & \vdots & \vdots & \vdots & . & . & \vdots \\ \vdots & . & . & \vdots & . & . & \vdots \\ \vdots & . & . & \vdots & . & . & \tilde{a}_{n-\beta,\beta} \\ \vdots & . & . & \vdots & . & / & 0 \\ \vdots & . & . & . & / & / & \vdots \\ \tilde{a}_{n,-\beta} & \dots & \dots & \tilde{a}_{n,0} & 0 & \dots & 0 \end{pmatrix} \quad n = (2\beta + 1) \qquad O(n)$$

$\tilde{a}_{is} = a_{i,i+s}$ for row $i = 1, \ldots, n \quad 1 \leq i + s \leq n$

$S \in [l_i, r_i] = [max\{-\beta, 1 - i\},\ min\{\beta,\ n - i\}]$

Therefore we get the inequality

$$1 - i \leq S \leq n - i, \quad -\beta \leq S \leq \beta, \quad 1 - S \leq i \leq n - S$$

e.g.

$$\begin{array}{lll} row & i = 1: & S \in [0, \beta] \\ & \vdots & \\ row & i = \beta + 1: & S \in [-\beta, \beta] \\ & \vdots & \\ & i = n - \beta: & S \in [-\beta, \beta] \\ & \vdots & \\ & i = n: & S \in [-\beta, 0] \end{array}$$

computation of matrix-vector-product $C = A \cdot b$ on vector processor

$$C_i \ = \ A_{ij} \cdot b \ = \ \sum_j a_{ij} b_j \ = \ \sum_{S=l_i}^{r_i} a_{i,i+S} . b_{\underbrace{i+S}_{j}} \ = \ \sum_{S=l_i}^{r_i} \tilde{a}_{i,S} \cdot b_{i+S}$$

for i = 1,. . .,n

Algorithm:

for s = - β : 1 : β
    for i = max{1-s, 1} : 1 : min {n-s,n}
        $c_i = c_i + \tilde{a}_{ij} b_{i+s}$
    end
end

} general triade (no SAXPY)

parallel computation:

$$\langle 1, n \rangle = \bigcup_{r=1}^{R} I_r$$

for i $\in I_r$

$$c_i = \sum_{s=l_i}^{r_i} \tilde{a}_{is} \cdot b_{i+s}$$

end

Processor P_r gets rows to index set $I_r := [m_r, M_r]$ to compute its part of C.

What part of vector b is necessary to process P_r ?

b_j for

$$j = i + s \geq m_r + l_{m_r} = m_r + max\{-\beta, 1 - m_r\} = max\{m_r - \beta, 1\}$$
$$j = i + s \leq M_r + r_{M_r} = M_r + min\{\beta, n - M_r\} = min\{M_r + \beta, n\}$$

Hence processor P_r ($\sim I_r$) needs
b_j for $j \in [max\{1, m_r - \beta\}, min\{n, M_r + \beta\}]$.

2.3 Analysis of the Matrix-Matrix-product

$$A = (a_{ij})_{\substack{i=1..n \\ j=1..m}} \qquad B = (b_{ij})_{\substack{i=1..m \\ j=1..q}} \qquad C = A \cdot B = (c_{ij})_{\substack{i=1..n \\ j=1..q}}$$

for i = 1..n, for j=1..q:

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj} = \begin{pmatrix} * & & * \\ \mathbf{a_{i1}} & \dots & \mathbf{a_{im}} \\ * & & * \end{pmatrix} \cdot \begin{pmatrix} * & \mathbf{b_{1j}} & * \\ & \vdots & \\ * & \mathbf{b_{mj}} & * \end{pmatrix} = \begin{pmatrix} * & & * \\ & \mathbf{c_{ij}} & \\ * & & * \end{pmatrix}$$

Algorithm 1 (ijk) - form:

```
for i = 1:n
    for j = 1:q
        for k = 1:m
              c_ij = c_ij + a_ik b_kj     }  DOT-product
        end                               }  c_ij = A_i. · B_.j
    end
end
```

All entries c_{ij} are fully computed, one after another. Access to A rowwise, to B columnwise

Algorithm 2 (jki) - form

```
for j = 1:q
    k = 1:m                                                       }
        for i = 1:n                   }  SAXPY                    }    GAXPY
              c_ij = c_ij + a_ik b_kj }  c_j = c_j + a_.k · b_kj  }  c_.j = Σ_k b_kj a_.k
        end                           }  vector c_j               }
    end                                                           }
end
```

c computed columnwise; access to A columnwise

Algorithm 3 (kji) - form

```
for k = 1:m
    for j = 1:q
        for i = 1:n
            c_ij = c_ij + a_ik b_kj
        end
    end
end
```

(inner loop over i: SAXPY; loop over j: NO GAXPY because different $c_{\cdot j}$)

There are computed intermediate values $c_{ij}^{(k)}$. Access to A columnwise.

	ijk	ikj	kij	jik	jki	kji
Access to A	row	—	—	row	column	column
Access to B	column	row	row	column	—	—
Computation of c	row	row	row	column	column	column
c_{ij}	direct	delayed	delayed	direct	delayed	delayed
vector operation	DOT	GAXPY	SAXPY	DOT	GAXPY	SAXPY
with vector length	m	q	q	m	m	m

usually GAXPY better;
longer vector length better;
choose the right access to A,B, depending on the storage.

Matrix-Matrix-product in parallel

$$\begin{aligned}\langle 1, n\rangle &= \bigcup_{r=1}^{R} I_r \\ \langle 1, m\rangle &= \bigcup_{s=1}^{S} K_s \\ \langle 1, q\rangle &= \bigcup_{t=1}^{T} J_t\end{aligned}$$

Distribute blocks relative to index sets $I_r,\ K_s,\ J_t$ to processor P_{rst}:

$$I_r \overset{K_s}{\begin{pmatrix} & \vdots & \\ \cdots & A_{rs} & \cdots \\ & \vdots & \end{pmatrix}} \cdot K_s \overset{J_t}{\begin{pmatrix} \cdots & B_{st} & \cdots \\ & \vdots & \end{pmatrix}} = I_r \overset{J_t}{\begin{pmatrix} & \vdots & \\ \cdots & c_{rt}^{(s)} & \cdots \\ & \vdots & \end{pmatrix}}$$

1. process P_{rst} : $c_{rt}^{(s)} = A_{rs} \cdot B_{st}$

small matrix-matrix-product all processors independently

2. sum:

$$c_{rt} = \sum_{s=1}^{S} c_{rt}^{(s)} \qquad \text{fan-in in S}$$

Special case S = 1:

$$I_r \begin{pmatrix} \overline{} \\ \overline{} \end{pmatrix} \cdot \begin{pmatrix} \Big| \; \overset{J_t}{} \; \Big| \end{pmatrix} = \begin{pmatrix} & \overset{J_t}{} & \\ & c_{rt} & \\ & & \end{pmatrix} I_r$$

Each process computes a block of c independently without communication. Each process needs full block of rows of A($\sim I_r$) and block of columns of B($\sim J_t$), to compute the block c_{rt}.

with $n \cdot q$ processor: each processor has to compute one DOT-product

$$c_{rt} = \sum_{k} a_{rk} b_{kt} \quad in\ O(m)$$

If we use more processors to compute all these DOT-products by fan-in, we can reduce the parallel complexity to O($\log m$).

3 Linear Equations with dense matrices

3.1 Gaussian Elimination: Basic facts

Linear equations

$$\begin{aligned} a_{11}x_1 + \ldots + a_{1n}x_n &= b_1 \\ &\vdots \\ a_{n1}x_1 + \ldots + a_{nn}x_n &= b_n \end{aligned}$$

$$\longrightarrow \underbrace{\begin{pmatrix} a_{11} & \ldots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \ldots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}}_{Ax\ =\ b} \qquad \text{A} = A^{(1)}$$

Solving triangular equations is easy, so we try, to transform the given system in triangular form:

$$\begin{matrix}(1)\\(2)\\ \\(n)\end{matrix}\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & & & \\ a_{n1} & & & a_{nn}\end{pmatrix} \qquad \begin{matrix}(2) \rightarrow (2) - \frac{a_{21}}{a_{11}}.(1) \\ \\ (n) \rightarrow (n) - \frac{a_{n1}}{a_{11}}.(1)\end{matrix}$$

$$\longrightarrow = A^{(2)} = \begin{pmatrix} a_{11}^{(2)} & a_{12}^{(2)} & \dots & a_{1n}^{(2)} \\ 0 & a_{22}^{(2)} & \dots & a_{2n}^{(2)} \\ \vdots & a_{32}^{(2)} & & \\ \vdots & & & \\ 0 & & & a_{nn}^{(2)}\end{pmatrix} \qquad \begin{matrix}(3) \rightarrow (3) - \frac{a_{32}}{a_{22}}.(2) \\ \\ (n) \rightarrow (n) - \frac{a_{n2}}{a_{22}}.(2)\end{matrix}$$

$$\longrightarrow = A^{(3)} = \begin{pmatrix} a_{11}^{(3)} & a_{12}^{(3)} & & \dots & a_{1n}^{(3)} \\ 0 & a_{22}^{(3)} & & \dots & a_{2n}^{(3)} \\ \vdots & 0 & a_{33}^{(3)} & & \\ \vdots & & & & \\ 0 & 0 & a_{n3}^{(3)} & \dots & a_{nn}^{(3)}\end{pmatrix} \rightarrow \dots \rightarrow A^{(n)} = \begin{pmatrix} a_{11} & \dots & & a_{1n} \\ 0 & \ddots & & \\ \vdots & & & \\ & \ddots & a_{n-1,n-1} & a_{n-1,n} \\ 0 & & 0 & a_{nn}\end{pmatrix}$$

upper triangular form

No pivoting (we assume $a_{kk}^{(k)} \neq 0$ for all k), we ignore the right hand side b.

Algorithm:

```
for k = 1:n-1
    for i = k+1:n
        l_ik = a_ik / a_kk
    end
    for i = k+1:n
        for j = k+1:n
            a_ij = a_ij - l_ik a_kj
        end
    end
end
```

Intermediate System $A^{(u)} = \begin{pmatrix} A_{11}^{(u)} & & & \\ 0 & \ddots & & \\ & 0 & A_{kk}^{(u)} & A_{kn}^{(u)} \\ & & \vdots & \\ 0 & 0 & A_{nk}^{(u)} & A_{nn}^{(u)}\end{pmatrix}$

Define matrix with entries l_{ik} from above algorithm.

$$L = \begin{pmatrix} 1 & 0 & & & 0 \\ l_{21} & 1 & \ddots & & \\ \vdots & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & 0 \\ l_{n,1} & \dots & \dots & l_{n,n-1} & 1 \end{pmatrix} \text{ and } L_k = \begin{pmatrix} 0 & & & & & \\ & 0 & 0 & & \mathbf{0} & \\ & & l_{k+1,k} & 0 & & \\ & \mathbf{0} & \vdots & & \ddots & \\ & & l_{n,k} & 0 & \dots & 0 \end{pmatrix}$$

Each Elimination step in Gaussian Elimination can be written in the form

$$A^{(k+1)} = (1 - L_k) \cdot A^{(k)} = A^{(k)} - L_k A^{(k)} \tag{3}$$

with $A^{(1)} = A$ and $A^{(n)} = U =$ upper triangular
$U = A^{(n)} = (1 - l_{n-1})A^{(n-1)} = \ldots = \underbrace{(1 - l_{n-1}) \cdots (1 - l_1)}_{\tilde{L}} A^{(1)} = \tilde{L} \cdot A$

with $\tilde{L} := (1 - l_{n-1}) \cdots (1 - l_1)$

$1 - l_j$ *lower triangular* $\Rightarrow \tilde{L}$ *lower triangular* $\Rightarrow \tilde{L}^{-1}$ *lower triangular*
$\Rightarrow A = \tilde{L}^{-1}U$ with $\tilde{L}^{-1}$ lower and U upper triangular.

Theorem 2 $\tilde{L}^{-1} = L$

Proof:

$$i \le j \Rightarrow 0 = l_i \cdot l_j = \begin{pmatrix} 0 & & & \\ & \ddots & & \\ & & 0 & \\ & & * & \\ & & \vdots & \\ & & * & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & & & \\ & \ddots & & \\ & & 0 & \\ & & * & \\ & & \vdots & \\ & & * & 0 \end{pmatrix}$$

(the arrows $i \downarrow$ and $j \downarrow$ mark the column of the nonzero entries in each factor)

Therefore $(1 + l_j)(1 - l_j) = 1 + l_j - l_j - l_j^2 = I$ and $(1 - l_j)^{-1} = 1 + l_j$
$\Rightarrow \tilde{L}^{-1} \left[(1 - l_{n-1}) \ldots (1 - l_1)\right]^{-1} = (1 - l_1)^{-1} \ldots (1 - l_{n-1})^{-1}$
$= (1 + l_1)(1 + l_2) \ldots (1 + l_{n-1}) = 1 + l_1 + l_2 + \ldots + l_{n-1} = L$
because e.g. $(1 + l_1).(1 + l_2) = 1 + l_1 + l_2 + \underbrace{l_1 l_2}_{=0} = 1 + l_1 + l_2$

Total: $A = L \cdot U$ with L lower and U upper triangular.

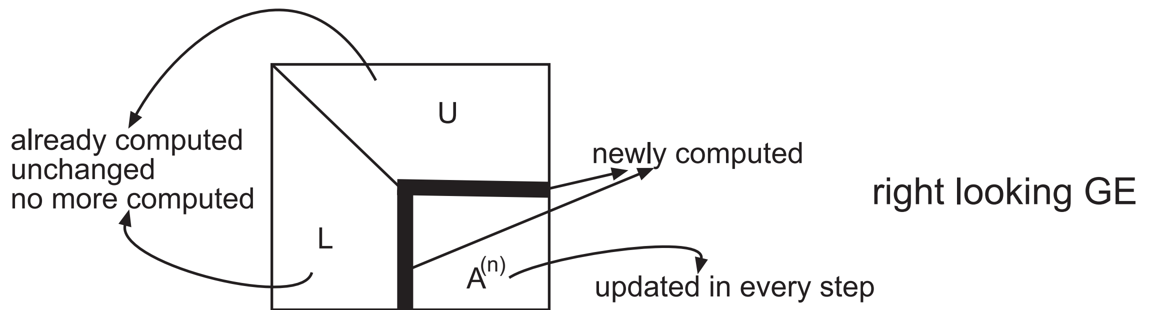
3.2 Vectorization of the Gaussian Elimination

(kij)-form (standard)

for k = 1:n-1
    for i = k+1:n
        $l_{i,k} = \frac{a_{ik}}{a_{kk}}$ } Vector operation $\alpha \cdot \vec{x}$
    end
    for i = k+1:n
        for j = k+1:n
            $a_{ij} = a_{ij} - l_{ik}a_{kj}$ } SAXPY in row $a_{i\cdot}$ and $a_{k\cdot}$
        end
    end } No GAXPY
end

U computed rowwise, columnwise

In the following, we want to interchange the kij-loops:

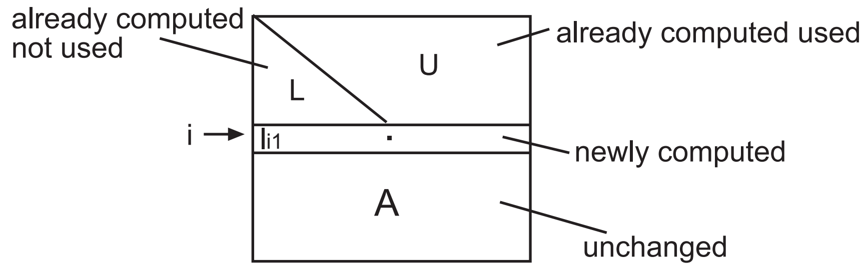


Necessary condition: $1 \leq k < i \leq n$
$1 \leq k < j \leq n$

(ikj)-form:

for i = 2:n
    for k = 1:i-1
        $l_{ik} = \frac{a_{ik}}{a_{kk}}$
        j = k+1:n
            $a_{ij} = a_{ij} - l_{ik}a_{kj}$
        end
    end } GAXPY in a_{ii}
end

compute l_{i1} by SAXPY
combine the 1st row and the i-th row, then compute $l_{12}, \ldots$ and so on.
L and U are computed rowwise.

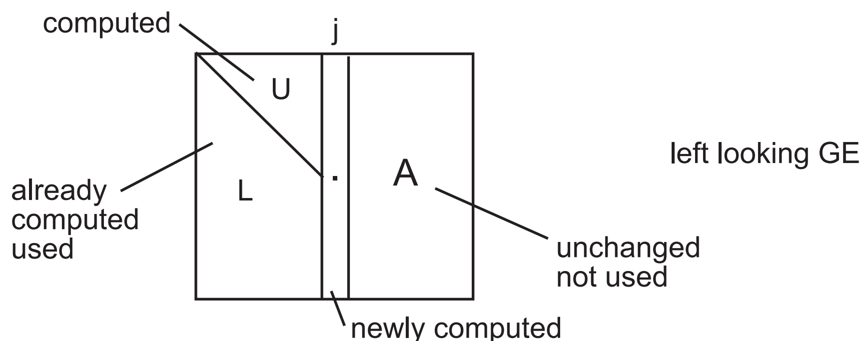


(ijk)-form:

```
for i = 2:n
    for j = 2:i
        l_{i,j-1} = a_{i,j-1} / a_{j-1,j-1}
        for k = 1:j-1
            a_ij = a_ij - l_ik a_kj        } DOT (upper left)
        end
    end
    for j = i+1:n
        for k = 1:i-1
            a_ij = a_ij - l_ik a_kj        } DOT (upper right)
        end
    end
end
```

(jki)-form

```
for j = 2:n
    for k = j:n
        l_{k,j-1} = a_{k,j-1} / a_{j-1,j-1}   } α · x⃗
    end
    for k = 1:j-1
        a_ij = a_ij - l_ik a_kj               } GAXPY in a_ij
    end
end
```



	kij	kji	ikj	ijk	jki	jik
Access to AU	row	column	row	column	column	column
Access to L	—	column	—	row	column	row
Computation of U	row	row	row	row	column	column
Computation of L	column	column	row	row	column	column
Vector Operation	SAXPY	SAXPY	GAXPY	DOT	GAXPY	DOT
Vector Length	$\frac{2}{3}n$	$\frac{2}{3}n$	$\frac{2}{3}n$	$\frac{1}{3}n$	$\frac{2}{3}n$	$\frac{1}{3}n$

Vector length = average of occurring vector lengths

3.3 Gaussian Elimination in Parallel: Blockwise GE

(better in environment) →

- (i) solve triangular system $L : U = A$ independently columns of U.
- (ii) $A_{22} - LU$ updating blocks (easy parallelize)
- (iii) small LU-decomposition

$$\begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \cdot \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}$$

$$= \begin{pmatrix} l_{11}u_{11} & l_{11}u_{12} & l_{11}u_{13} \\ l_{21}u_{11} & l_{21}u_{12} + l_{22}u_{22} & l_{21}u_{13} + l_{22}u_{23} \\ l_{31}u_{11} & l_{31}u_{12} + l_{32}u_{22} & * \end{pmatrix}$$

Different ways of computing L and U, depending on ordering:
→ different algorithm

3.3.1 Crout method

$$\begin{pmatrix} \mathbf{l_{11}} & 0 & 0 \\ \mathbf{l_{21}} & \mathit{l_{22}} & 0 \\ \mathbf{l_{31}} & \mathit{l_{32}} & * \end{pmatrix} \cdot \begin{pmatrix} \mathbf{u_{11}} & \mathbf{u_{12}} & \mathbf{u_{13}} \\ 0 & \mathit{u_{22}} & \mathit{u_{23}} \\ 0 & 0 & * \end{pmatrix}$$

In **Bold**: already computed
In *italics*: has to be computed in this step

$$\left(\begin{array}{c|c} l_{22}u_{22} & l_{22}u_{23} \\ l_{32}u_{22} & * \end{array} \right) \stackrel{!}{=} \left(\begin{array}{c|c} A_{22} - l_{21}u_{12} & A_{23} - l_{21}u_{13} \\ A_{32} - l_{31}u_{12} & * \end{array} \right) = \begin{pmatrix} \hat{A}_{22} & \hat{A}_{23} \\ \hat{A}_{32} & * \end{pmatrix}$$

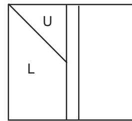
(1) $\begin{pmatrix} l_{22} \\ l_{32} \end{pmatrix} U_{22} = \begin{pmatrix} \hat{A}_{22} \\ \hat{A}_{32} \end{pmatrix}$ by small LU-decomposition gives $l_{22}, l_{32}, and\, U_{22}$

(2) $l_{22}u_{23} = \hat{A}_{23} \Rightarrow$ by solving triangular system in l_{22}

in total: $\left(\begin{array}{cc|c} l_{11} & & \\ l_{21} & l_{22} & \\ l_{31} & l_{32} & l_{33} \end{array} \right) \cdot \left(\begin{array}{ccc} U_{11} & U_{12} & U_{13} \\ & U_{22} & U_{23} \\ \hline & & U_{33} \end{array} \right) = A$

Put the computed parts in the first row/column blocks of L and U
Split l_{33} and U_{33} in new parts $l_{22}, l_{32}, U_{22}, U_{23}$ and repeat.

3.3.2 left looking GE:



$$\begin{pmatrix} \mathbf{l_{11}} & 0 & 0 \\ \mathbf{l_{21}} & l_{22} & 0 \\ \mathbf{l_{31}} & l_{32} & l_{33} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{u_{11}} & \mathit{u_{12}} & u_{13} \\ & \mathit{u_{22}} & u_{23} \\ & & u_{33} \end{pmatrix} = A$$

In **Bold**: already computed
In *italics*: has to be computed in this step

equations: $l_{11}u_{12} = A_{12}$ can be solved by triangular gives u_{12}

Compute $\begin{pmatrix} \hat{A}_{22} \\ \hat{A}_{32} \end{pmatrix} = \begin{pmatrix} A_{22} \\ A_{32} \end{pmatrix} - \begin{pmatrix} l_{21} \\ l_{31} \end{pmatrix} U_{12}$ by matrix multiplication

and $\begin{pmatrix} l_{22} \\ l_{32} \end{pmatrix} U_{22} = \begin{pmatrix} \hat{A}_{22} \\ \hat{A}_{32} \end{pmatrix}$ small LU-decomposition $\rightarrow l_{22}, l_{32}, U_{22}$

3.3.3 Right looking / Gaussian Elimination standard

$$\begin{pmatrix} \mathbf{l_{11}} & & \\ \mathit{l_{21}} & \mathit{l_{22}} & \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{u_{11}} & \mathit{u_{12}} & u_{13} \\ & \mathit{u_{22}} & u_{23} \\ & & u_{33} \end{pmatrix} = A$$

In **Bold**: already computed
In *italics*: has to be computed in this step

$A_{11} = l_{11}u_{11}$ (small LU-decomposition) with equations:

$l_{21}u_{11} = A_{21} \Rightarrow l_{21}$; $l_{11}u_{12} = A_{12} \Rightarrow u_{12}$ triangular solve

$l_{22}u_{22} = A_{22} - l_{21}u_{12} = \hat{A}_{22}$ by LU-decomposition of $\hat{A}_{22}$

In comparison, all variants have nearly the same efficiency in parallel, flops in Matrix-Matrix-Multiplication, triangular solve and LU-decomposition.

3.4 QR-Decomposition with Householder matrices

3.4.1 QR-decomposition

Similar to LU-decomposition (numerically not stable) by Gaussian-Elimination.
We are interested in A = QR with Q orthogonal and R upper triangular.
$b = Ax = QRx \quad \Leftrightarrow \quad Rx = Q^T b$ for solving linear system.
QR has advantages for ill-conditioned A.

Application for overdetermined systems $\boxed{\mathbf{A}} \cdot \boxed{x} \overset{!}{=} \boxed{\mathbf{b}}$

Ax = b has no solution. best approximate solution by solving
$\min\limits_x \|Ax - b\|_2^2 = \min\limits_x (x^T A^T Ax - 2x^T A^T b + b^T b)$
gradient equal zero leads to $\mathbf{A^T A x = A^T b}$ (normal equation).
$A^T A$ has a larger condition number than A.
Advantages of QR-decomposition:

$$A = QR \quad , \quad R = \begin{pmatrix} R_1 \\ \hline 0 \end{pmatrix} \quad , \quad cond(R_1) = cond(A)$$

$$\begin{aligned} A^T Ax = A^T b \quad & \Leftrightarrow \quad (QR)^T QRx = (QR)^T b \quad \Leftrightarrow \quad R^T Rx = R^T \underbrace{Q^T b}_{\hat{b}} \\ & \Leftrightarrow \quad \begin{pmatrix} R_1^T & 0 \end{pmatrix} \begin{pmatrix} R_1 \\ 0 \end{pmatrix} x = \begin{pmatrix} R_1^T & 0 \end{pmatrix} \hat{b} \quad \Leftrightarrow \quad R_1^T R_1 x = \begin{pmatrix} R_1^T & 0 \end{pmatrix} \begin{pmatrix} \hat{b}_1 \\ \hat{b}_2 \end{pmatrix} \\ & \Leftrightarrow \quad R_1^T R_1 x = R_1^T \hat{b}_1 \quad \Leftrightarrow \quad \mathbf{R_1 x = \hat{b}_1} \end{aligned}$$

3.4.2 Householder method for QR

u vector $\in \mathbf{R^n}$ with length 1, $\|u\|_2 = 1$
$H := 1 - 2uu^T$ is called Householder matrix (rank-1-perturbation of identity)
H is orthogonal ($H^T H = 1$) ; $H = H^T$:
$H^T H = H^2 = (1 - 2uu^T)(1 - 2uu^T) = 1 - 2uu^T - 2uu^T + 4u \underbrace{u^T u}_{1} u^T = I$
First step: use H to transform the first column of A in upper triangular form:

$$H_1 A = (1 - 2u_1 u_1^T)(a_1|...) = (a_1 - 2(u_1^T a_1)u_1| * ..*) \stackrel{!}{=} \left(\begin{array}{c|c} \alpha & \\ 0 & \\ \vdots & ** \\ 0 & \end{array} \right)$$

Hence we have to find u_1 of length 1 with $\mathbf{a_1 - 2(u_1^T a_1) u_1} \stackrel{!}{=} \alpha \mathbf{e_1}$

$$H_1 \text{ is orthogonal, therefore } \|a_1\|_2 = \left\| \begin{pmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{pmatrix} \right\| = |\alpha|$$

We can set $\alpha = \|a_1\|$, therefore $u_1 = \frac{a_1 - \|a_1\|_2 e_1}{2(u_1^T a_1)} = \frac{a_1 - \|a_1\|_2 e_1}{\|a_1 - \|a_1\|_2 e_1\|_2}$

$$H_1 A_1 = (1 - 2u_1 u_1^T) A = \left(\begin{array}{c|c} \|a_1\| & * \;...\; * \\ \hline 0 & \\ \vdots & A_2 \\ 0 & \end{array} \right) \qquad V_1 := u_1$$

Apply the same procedure on A_2:

$$H_2 A_2 = (1 - 2u_2 u_2^T) A_2 = \left(\begin{array}{c|c} * & \\ \hline 0 & \\ \vdots & A_3 \\ 0 & \end{array} \right)$$

$(1 - 2u_2 u_2^T) \longrightarrow$ dimension n-1
extend u_2 to vector of length n : $v_2 := \begin{pmatrix} 0 \\ u_2 \end{pmatrix}$,

$$\text{Hence } H_2 H_1 A = (1 - 2v_2 v_2^T)(1 - 2v_1 v_1^T) A = \begin{pmatrix} * & & \\ 0 & * & \\ & 0 & \\ \vdots & & \boxed{A_3} \\ 0 & 0 & \end{pmatrix}$$

Total: $\underbrace{H_{n-1}\cdots H_2 H_1}_{Q^T} A = R =$ upper triangular

$A = QR \quad$ with $\quad Q = (H_{n-1}\cdots H_2 H_1)^T = (H_1 \cdots H_{n-1})$

3.4.3 Householder method in parallel

Idea: Compute $u_1 \ldots u_k$, but application of $H_k \ldots H_1 A$ in *blocked* form for elimination of first k columns

Question: What is the structure of $H_k \ldots H_i =: V$

$A = (\overbrace{A_1}^{k} | A_2) \stackrel{?}{=} QR$

compute u_1 , $H_1 = I - 2u_1 u_1^T$, $H_1 A_1$

compute u_2 , $H_2 = I - 2u_2 u_2^T$, $H_2(H_1 A_1)$ usw...

Theorem 3 $H_k \ldots H_i = (1-2v_k v_k^T)\ldots(1-2v_i v_i^T) = I-(v_k \ldots v_i)T_i \begin{pmatrix} v_k^T \\ \vdots \\ v_i^T \end{pmatrix}$

with T_i upper triangular matrix

Proof by Induction:

$$\underbrace{\left[(1-2v_k v_k^T)\ldots(1-2v_i v_i^T)\right]}_{Assumption}(1-2v_{i-1}v_{i-1}^T)$$

$$= \left[I - (v_k \ldots v_i)T_i \begin{pmatrix} v_k^T \\ \vdots \\ v_i^T \end{pmatrix} \right](1-2v_{i-1}v_{i-1}^T)$$

$$= I - 2v_{i-1}v_{i-1}^T - (v_k \ldots v_i)T_i \begin{pmatrix} v_k^T \\ \vdots \\ v_i^T \end{pmatrix} + 2(v_k \ldots v_i)\underbrace{T_i \begin{pmatrix} v_k^T v_{i-1} \\ \vdots \\ v_i^T v_{i-1} \end{pmatrix}}_{y} v_{i-1}^T$$

$$= I - (v_k \ldots v_i v_{i-1}) \left(\begin{array}{c|c} T_i & \text{-2y} \\ \hline 0 & 2 \end{array} \right) \begin{pmatrix} v_k^T \\ \vdots \\ v_i^T \\ v_{i-1}^T \end{pmatrix}$$

Computation of $\underline{H_k \ldots H_i A}$ as $(I-YTY^T)A = A-YT(Y^T A) = \left(\begin{array}{c|c} \text{R} & * \\ \hline 0 & \tilde{A} \end{array} \right)$

with $y = (u_1, \ldots, u_k)$ and then repeat with $\tilde{A}$.

4 Linear Equations with sparse matrices

4.1 General properties of sparse matrices

Full n × n matrix: $\left.\begin{array}{c} O(n^2)\text{storage} \\ O(n^3)\text{solution} \end{array}\right\}$ too costly

Formulate the given problem such that the resulting linear system is sparse
$O(n)$ storage $O(n)$ solution?

example: tridiagonal: most of the entries are zero
Example matrix:

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 6 & 0 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 & 0 \\ 0 & 0 & 0 & 0 & 12 \end{pmatrix} \text{; n = 5, nnz (number of nonzero entries) = 12}$$

4.1.1 Storage in coordinate form

values	AA	12	9	7	5	1	2	11	3	6	4	8	10
row	JR	5	3	3	2	1	1	4	2	3	2	3	4
column	JC	5	5	3	4	1	4	4	1	1	2	4	3

Superfluous information:
(storage nnz floating point numbers, 2nnz+2 integer numbers)

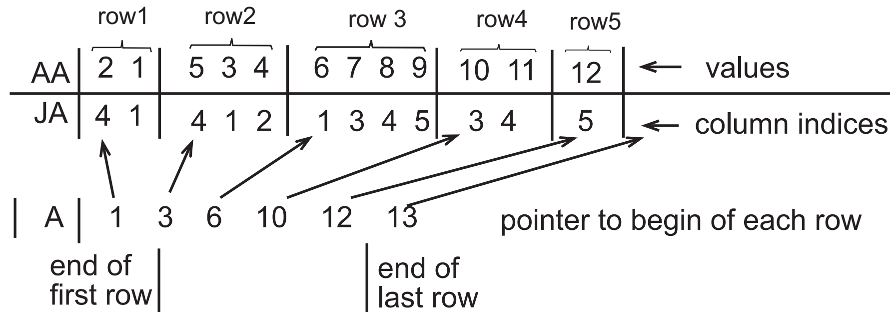
Computation of $C = Ab$

for $j = 1 : nnz(A)$
$$C_{JR(j)} = C_{JR(j)} + \underbrace{AA_{(j)}}_{a_{JR(j),JC(j)}} \; b_{JC(j)};$$
end

indirect addressing (indexing) no c and b → jumping in memory (Disadvantage)

Advantage: does not prefer rows or columns.

4.1.2 Compressed Sparse Row Format: CSR



(nnz floating point numbers ; nnz+n+3 integer numbers)

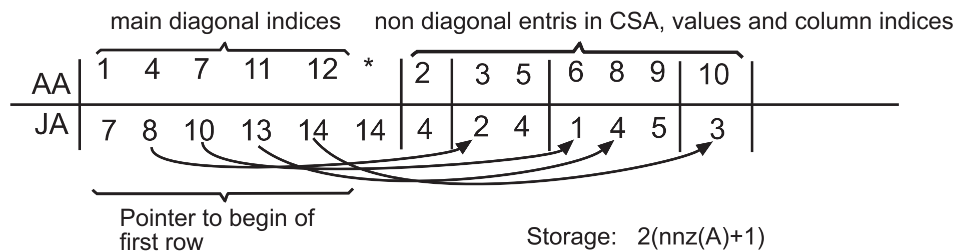
$C = Ab$:

for $i = 1 : n$
  for $IA(i) : IA(i+1) - 1$
    $C(i) = C(i) + AA(j)b(JA(j))$
  end
end

only indirect addressing and jumps in b. *Compressed Sparse Column format*

4.1.3 Improving CSR by extracting the main diagonal entries

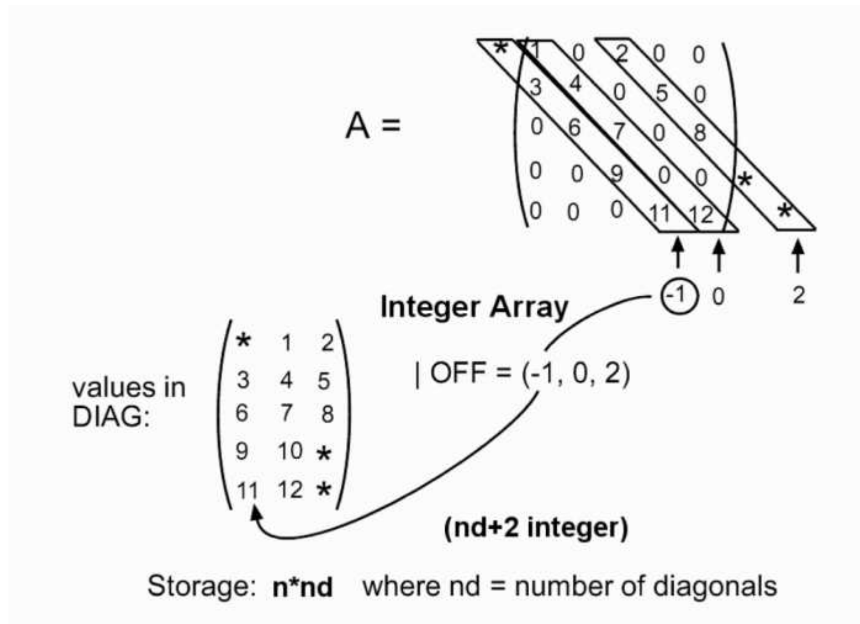
(nnz+1 floating point numbers ; nnz+1+2 integer numbers)



for $i = 1 : n$
  $C(i) = AA(i)b(i)$
  for $JA(i) : JA(i+1) - 1$
    $c(i) = c(i) + AA(j)b(JA(j))$
  end
end

4.1.4 Diagonalwise storage, e.g. for band matrices

example:



only efficient for band matrices

4.1.5 rectangular, rowwise storage scheme by compressing from the right

$$\left.\begin{array}{ccccc} 1 & 0 & 2 & 0 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 0 & 6 & 7 & 0 & 8 \\ 0 & 0 & 9 & 10 & 0 \\ 0 & 0 & 0 & 11 & 12 \end{array}\right| \longleftarrow \text{ gives COEF (values)} = \begin{pmatrix} 1 & 2 & 0 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & 0 \\ 11 & 12 & 0 \end{pmatrix}$$

$$\text{JCOEF (columnination)} = \begin{pmatrix} 1 & 3 & * \\ 1 & 2 & 4 \\ 2 & 3 & 5 \\ 3 & 4 & * \\ 4 & 5 & * \end{pmatrix}$$

storage: $\underbrace{n}_{=5} * \underbrace{\text{nnz of longest row of A}}_{nl=3}$

$C = Ab \quad ; \quad C = 0$

for $i = 1 : n$

  for $j = 1 : nl$

    $C(i) = C(i) + COEFF(i,j) * b(JCOEFF(i,j))$

  end

end

ELLPACK

4.1.6 Jagged diagonal form

First step: Sort rows after their length:

$$A = \begin{pmatrix} 1 & 0 & 2 & 0 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 0 & 6 & 7 & 0 & 8 \\ 0 & 0 & 9 & 10 & 0 \\ 0 & 0 & 0 & 11 & 12 \end{pmatrix} \longrightarrow PA = \begin{pmatrix} 3 & 4 & 0 & 5 & 0 \\ 0 & 6 & 7 & 0 & 8 \\ 1 & 0 & 2 & 0 & 0 \\ 0 & 0 & 9 & 10 & 0 \\ 0 & 0 & 0 & 11 & 12 \end{pmatrix}$$

Lenght 3

Length 2

Storage for PA in the form:

DJ values: $\underbrace{3\ 6\ 1\ 9\ 11}_{\text{first jagged diagonal}} \mid \underbrace{4\ 7\ 2\ 10\ 12}_{\text{second}} \mid 5\ 8 \mid$

Column indices:

JDIAG:	1	2	1	3	4	2	3	3	4	5	4	5	
IDIAG:	1	2	3	4	5	6	7	8	9	10	11	12	13

$C = Ab \quad ; \quad C = 0$

$j = 1 : NDIAG$

  $for\ i = 1 : \overbrace{IDIAG(j+1) - IDIAG(j)}^{length\ of\ j\ jagged\ diagonals}$

    $C(i) = C(i) + DJ(\overbrace{\underbrace{IDIAG(j)}_{similar\ to\ SAXPY} + i - 1}^{k}) b(JDIAG(\overbrace{IDIAG(j) + i - 1}^{k}))$

  end

end

Operations on local block data!

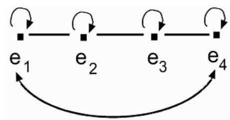
4.2 Sparse Matrices and Graphs

4.2.1 $A = A^T > 0$ (n × n - matrix) symmetric, define Graph G(A):

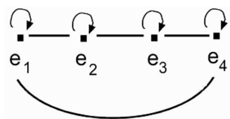
Knots, vertices: $e_1, \ldots, e_n$; edges $(e_i, e_j) \Leftrightarrow a_{ij} \neq 0$

example: $$A = \begin{pmatrix} * & * & 0 & * \\ * & * & * & 0 \\ 0 & * & * & * \\ * & 0 & * & * \end{pmatrix}$$

$\rightarrow G(A) =$



<u>undirected graph:</u>

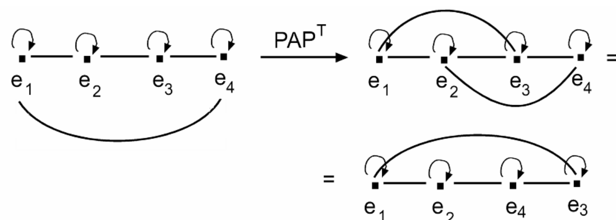


Graph G(A) has adjacency matrix

$$A(G(A)) = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$ has exactly the structure of A.

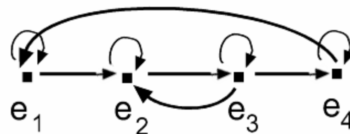
Symmetric permutation PAP^T, by permuting row and columns of A in the same way: renumbering of the knots

(renumbering $3 \leftrightarrow 4$ means, that $r_3 \leftrightarrow r_4$ and $c_3 \leftrightarrow c_4$)



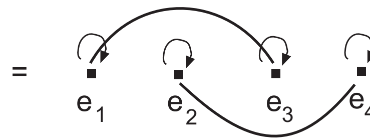
4.2.2 A non symmetric: directed graph

$$\begin{pmatrix} * & * & 0 & 0 \\ 0 & * & * & 0 \\ 0 & * & * & * \\ * & 0 & 0 & * \end{pmatrix} \longrightarrow$$



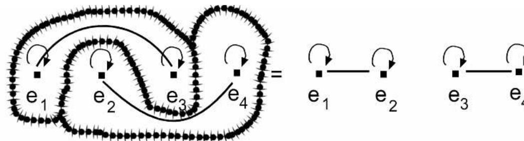
"good" sparsity pattern: Block Diagonal:

Example: $A = \begin{pmatrix} * & 0 & * & 0 \\ 0 & * & 0 & * \\ * & 0 & * & 0 \\ 0 & * & 0 & * \end{pmatrix} \longrightarrow$



Graph splits into two subgraphs; use permutation that groups together edges in the same subgraph: $2 \Leftrightarrow 3$

new G(A):



$$PAP^T = \begin{pmatrix} * & * & 0 & 0 \\ * & * & 0 & 0 \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{pmatrix} = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix} \text{ block diagonal}$$

Reduce the solution of the large given matrix to the solution of small block parts. The block pattern is not disturbed in Gauss-Elimination.

$$\begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}^{-1} = \begin{pmatrix} A_1^{-1} & 0 \\ 0 & A_2^{-1} \end{pmatrix}$$

<u>Band Matrix</u>: $A = \begin{pmatrix} a_{11} & \dots & a_{1p} & 0 & & 0 \\ \vdots & \ddots & & \ddots & & \vdots \\ a_{q1} & & \ddots & & \ddots & 0 \\ 0 & \ddots & & \ddots & & \vdots \\ \vdots & & \ddots & & \ddots & \vdots \\ 0 & & 0 & & \dots & a_{nn} \end{pmatrix}$

Gaussian Elimination without pivoting maintains this pattern.

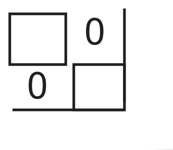
cols: O(n · pq) and $A = LU$ with

$$L = \begin{pmatrix} l_{11} & 0 & & & & 0 \\ \vdots & \ddots & & & & \\ l_{q1} & & \ddots & & & \\ 0 & \ddots & & \ddots & & \\ & & \ddots & & \ddots & 0 \\ 0 & & 0 & & \dots & l_{nn} \end{pmatrix} \text{ and } U = \begin{pmatrix} u_{11} & \dots & u_{1p} & 0 & & 0 \\ 0 & \ddots & & \ddots & & \\ & & \ddots & & \ddots & 0 \\ & & & \ddots & & \\ & & & & \ddots & \vdots \\ 0 & & & & 0 & u_{nn} \end{pmatrix}$$

with pivoting u will have a larger bandwidth

Similarly: A= structure is preserved by GE

4.2.3 Dissection form preserved during GE



(no fill-in GE)

Schur Complement for Block Matrices: Reduce to smaller matrices:

$$\begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix} \cdot \begin{pmatrix} B_1^{-1} & D \\ 0 & S^{-1} \end{pmatrix} = \begin{pmatrix} I & B_1 D + B_2 S^{-1} \\ B_3 B_1^{-1} & B_3 D + B_4 S^{-1} \end{pmatrix} \stackrel{!}{=} \begin{pmatrix} I & 0 \\ * & I \end{pmatrix}$$

Therefore $B_1 D + B_2 S^{-1} \stackrel{!}{=} 0 \Longrightarrow D = -B_1^{-1} B_2 S^{-1}$

and $B_3 D + B_4 S^{-1} \stackrel{!}{=} I \Longrightarrow I = -B_3 B_1^{-1} B_2 S^{-1} + B_4 S^{-1}$

$\Longrightarrow I = (B_4 - B_3 B_1^{-1} B_2) S^{-1}$

$\underline{\Longrightarrow \; S = B_4 - B_3 B_1^{-1} B_2}$ (Schur Complement)

$$\begin{pmatrix} B_1 & B_2 \\ B_3 & B_4 \end{pmatrix} = \begin{pmatrix} I & 0 \\ B_3 B_1^{-1} & I \end{pmatrix} \cdot \begin{pmatrix} B_1 & B_2 \\ 0 & S \end{pmatrix}$$

Instead of solving LE in B, we have to solve small systems in B_1 and S

Application in Dissection form:

$$\begin{pmatrix} A_1 & 0 & F_1 \\ 0 & A_2 & F_2 \\ G_1 & G_2 & A_3 \end{pmatrix}$$

Schur complement relative to $\begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$:

$$\begin{aligned} S &= A_3 - \begin{pmatrix} G_1 & G_2 \end{pmatrix} \begin{pmatrix} A_1^{-1} & 0 \\ 0 & A_2^{-1} \end{pmatrix} \begin{pmatrix} F_1 \\ F_2 \end{pmatrix} \\ &= A_3 - G_1 A_1^{-1} F_1 - G_2 A_2^{-1} F_2 \end{aligned}$$

Linear Equation in Dissection form:

$$\begin{pmatrix} A_1 & 0 & F_1 \\ 0 & A_2 & F_2 \\ G_1 & G_2 & A_3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \Longrightarrow \begin{aligned} A_1 x_1 + F_1 x_3 &= b_1 \\ A_2 x_2 + F_2 x_3 &= b_2 \\ G_1 x_1 + G_2 x_2 + A_3 x_3 &= b_3 \end{aligned}$$

$$\Longrightarrow \begin{aligned} x_1 &= A_1^{-1} b_1 - A_1^{-1} F_1 x_3 \\ x_2 &= A_2^{-1} b_2 - A_2^{-1} F_2 x_3 \end{aligned}$$

$\Longrightarrow (G_1 A_1^{-1} b_1 - G_1 A_1^{-1} F_1 x_3) + (G_2 A_2^{-1} b_2 - G_2 A_2^{-1} F_2 x_3) + A_3 x_3 = b_3$

$\Longrightarrow (A_3 - G_1 A_1^{-1} F_1 - G_2 A_2^{-1} F_2) x_3 = b_3 - G_1 A_1^{-1} b_1 - G_2 A_2^{-1} b_2$

$\boxed{\mathbf{S x_3 = \hat{b_3}}}$

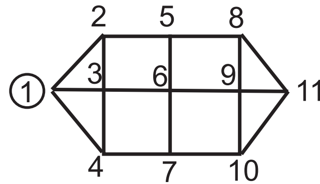
1. Compute S by using A_1^{-1} and A_2^{-1}
2. Solve $Sx_3 = b_3$
3. Compute x_1 and x_2 by using A_1^{-1} and A_2^{-1}

Sometimes S is full or too expensive to compute. Then use iterative method for $Sx_3 = \hat{b_3}$, that uses only s*vector, which can be computed easily with $F_1,\ F_2,\ G_1,\ G_2$ and $A_1^{-1},\ A_2^{-1}$.

4.3 Reordering

4.3.1 Smaller Bandwidth by Cuthill Mckee-Algorithm

Given sparse matrix A, $G(A)$



Define level sets:

$S_1 = \{1\}$

$S_2 =$ set of new edges connected to S_1 by vertex $\{2,\ 3,\ 4\}$

$S_3 =$ set of new edges connected to S_2 by vertex $\{5,\ 6,\ 7\}$

$S_4 = \{8,\ 9,\ 10\}$

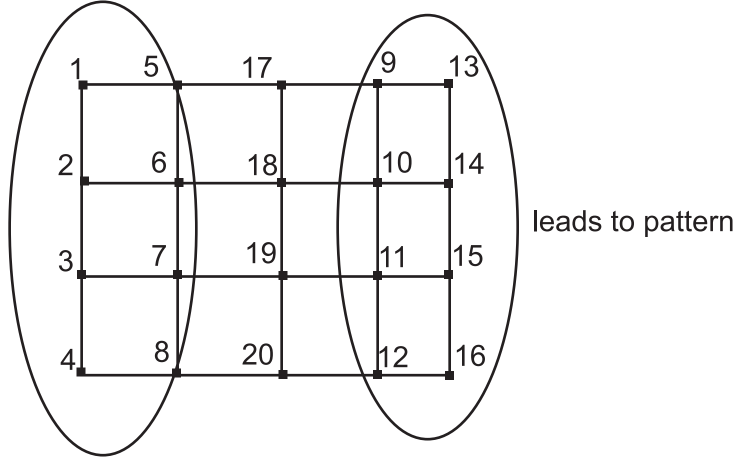
$S_5 = \{11\}$

Starting from one chosen vertex, according distance to the start knot.
First edge in S_1 gets number 1.
In each level set we sort and order the knots such that the first group of entries in S_i are the neighbours of the first entry in S_{i-1}, and the second group of entries in S_{i+1} are the neighbours of the second entry in S_i, and so on.

$$
\begin{pmatrix}
1 & * & * & * & & & & & & & \\
* & 2 & * & & * & & & & & & \\
* & * & 3 & * & & * & & & & & \\
* & & * & 4 & & & * & & & & \\
& * & & & 5 & * & & * & & & \\
& & * & & * & 6 & * & & * & & \\
& & & * & & * & 7 & & & * & \\
& & & & * & & & 8 & * & & * \\
& & & & & * & & * & 9 & * & * \\
& & & & & & * & & * & 10 & \\
& & & & & & & * & * & & 11
\end{pmatrix}
$$

often Cuthill McKee-Algorithm ordering is reversed: Reverse Cuthill McKee

4.3.2 Dissection Reordering



$$
\left(\begin{array}{cccccccc|cccccccc|cccc}
1 & * & & & * & & & & & & & & & & & & & & & \\
* & 2 & * & & & * & & & & & & & & & & & & & & \\
& * & 3 & * & & & * & & & & & & & & & & & & & \\
& & * & 4 & & & & * & & & & & & & & & & & & \\
* & & & & 5 & * & & & & & & & \mathbf{0} & & & & * & & & \\
& * & & & * & 6 & * & & & & & & & & & & & * & & \\
& & * & & & * & 7 & * & & & & & & & & & & & * & \\
& & & * & & & * & 8 & & & & & & & & & & & & * \\
\hline
& & & & & & & & 9 & * & & & * & & & & * & & & \\
& & & & & & & & * & 10 & * & & & * & & & & * & & \\
& & & & & & & & & * & 11 & * & & & * & & & & * & \\
& & & & & & & & & & * & 12 & & & & * & & & & * \\
& & & \mathbf{0} & & & & & * & & & & 13 & * & & & & & & \\
& & & & & & & & & * & & & * & 14 & * & & & & & \\
& & & & & & & & & & * & & & * & 15 & * & & & & \\
& & & & & & & & & & & * & & & * & 16 & & & & \\
\hline
& & & & * & & & & * & & & & & & & & 17 & & & \\
& & & & & * & & & & * & & & & & & & & 18 & & \\
& & & & & & * & & & & * & & & & & & & & 19 & \\
& & & & & & & * & & & & * & & & & & & & & 20
\end{array}\right)
$$

$$
= \begin{pmatrix} \boxed{A_1} & 0 & \boxed{F_1} \\ 0 & \boxed{A_2} & \boxed{F_2} \\ \boxed{G_1} & \boxed{G_2} & \boxed{A_3} \end{pmatrix}
$$

4.3.3 Algebraic pivoting: during GE

(Numerical pivoting: choose largest $a_{ij} \longrightarrow a_{kk}$ as pivot element)

Algebraic pivoting: choose largest a_{ij} from sparse row/column
$\longrightarrow a_{kk}$ as pivot element (small fill-in during GE-step)

Minimum degree re-ordering for $A = A^T > 0$

<u>first step</u>:

define $r_j = \#$ entries in row j (non-zero)

in the G(A) = # edges connected with vertex j

choose i such that $r_i = \overset{min}{j} r_j$ (nearly empty now)

choose a_{ii} pivot element $i \leftrightarrow 1$ by symmetric permutation

Do the elimination step in GE

reduce the matrix by one

Repeat.

Generalization to nonsymmetric case: Markowitz-Criterion

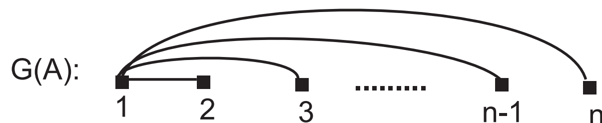
define $r - j = \#$ entries in row j (non-zero)

$c_k = \#$ entries in column k (non-zero)

minimizes: $\min_{j,k}(r_j - 1)(c_k - 1)$ choose a_{jk} as pivot element.

Apply permutation to put $a_{j,k}$ in diagonal position
In practise: mixtures of algebraic and numerical pivoting:
include a condition, that $|a_{i,s}|$ should be not too small!

$$\text{Example: } \begin{pmatrix} * & * & \cdots & & * \\ * & * & & 0 & \\ \vdots & & \ddots & & \\ & 0 & & & \\ * & & & & * \end{pmatrix} \xrightarrow{GE} \begin{pmatrix} * & * & \cdots & & * \\ 0 & \boxed{\begin{matrix} * & * & \cdots & * \\ & * & & * \\ \vdots & & \ddots & * \\ & * & * & * \end{matrix}} \\ \vdots & \\ \vdots & \\ 0 & \end{pmatrix} -- full$$

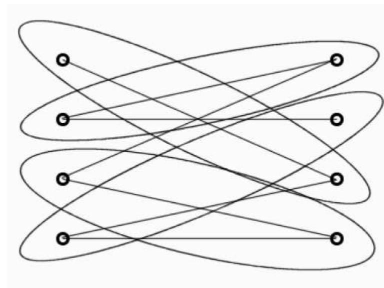


Cuthill-Mckee with starting edge $\{1\}:\ S_1=\{1\},\ S_1=\{1,\ldots,n\}$
given no improvement with start $\{2\}:\ S_1=\{1\},\ S_2=\{1\},\ S_3=\{2,\ldots,n\}$
...

Permutation such that smallest bandwidth also not very helpful.

Matching: set of edges, for each row/column index there is exactly one edge.

Matching gives a permutation of the rows $\Rightarrow$ nonzero diagonal entries.



Example: $\begin{pmatrix} 0 & 0 & * & 0 \\ * & * & 0 & 0 \\ * & 0 & 0 & 0 \\ 0 & * & * & * \end{pmatrix}$ $\quad \omega(\pi)=\sum_{i,j\ nonzero} log|a_{ij}|$

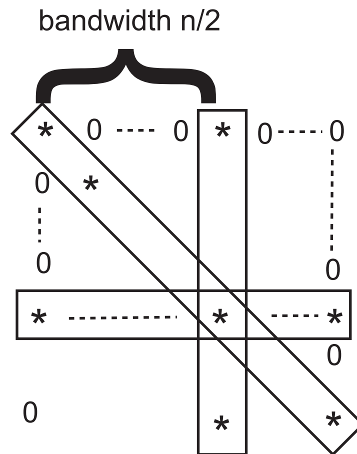
move here for example (1,3) to (3,3) and (2,1) to (1,1)

Perfect matching, maximizing $\omega(\pi)$ heuristic methods to get approximal solutions. For symmetric matrix we need a symmetric permutation PAP^T.

low permutation $\leftrightarrow$ perfect matching

$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix} \quad 1\rightarrow 2\rightarrow 3\rightarrow 1$

$\begin{pmatrix} 1 & 3 \end{pmatrix}\begin{pmatrix} 3 & 2 \end{pmatrix}:\begin{pmatrix} 1 & 3 \\ 3 & 1 \end{pmatrix}\begin{pmatrix} 3 & 2 \\ 2 & 3 \end{pmatrix} \quad 1\rightarrow 3\rightarrow 2\ ,\ 3\rightarrow 1\ ,\ 2\rightarrow 3$



Minimum Degree: is very good
Choose edge 1 with degree n-1
Therefore replace 1 by 2 with degree 1

$$\longrightarrow \begin{pmatrix} * & * & 0 & & & 0 \\ * & * & \dots & \dots & & * \\ 0 & & * & 0 & & 0 \\ & & 0 & \ddots & & \\ & & \vdots & & \ddots & 0 \\ 0 & * & 0 & & 0 & * \end{pmatrix} \xrightarrow{GE} \left(\begin{array}{c|cccccc} * & & * & \dots & & & * \\ \hline 0 & * & * & \dots & & & * \\ 0 & * & * & 0 & & & 0 \\ \vdots & & & 0 & & & \\ \vdots & \vdots & & & & \ddots & 0 \\ 0 & * & 0 & & & 0 & * \end{array}\right)$$

Next pivot 3, and so on ; works in $O(n)$

$$\text{Global reordering:} \begin{pmatrix} * & * & 0 & & & 0 \\ * & * & \dots & \dots & & * \\ 0 & & * & 0 & & 0 \\ & & 0 & \ddots & & \\ & & \vdots & & \ddots & 0 \\ 0 & * & 0 & & 0 & * \end{pmatrix} \longrightarrow \left(\begin{array}{cccc|c} * & & & & * \\ & * & & & \\ & & \ddots & & \\ & & & * & * \\ \hline ** & & * & & * \end{array}\right)$$

be permutation $1 \leftrightarrow n$; GE in $O(n)$

Change the numbering such that indices in a 2×2 permutation have subsequent numbers: Apply symmetric permutation.
The <u>large entries</u> appear in 2×2 <u>diagonal</u> blocks.

4.4 Gaussian Elimination in Graph

$A = A^T > 0$ symmetric positive definite
No need for numerical pivoting
Example G(A):

4 8 12
7
3 11
6
2 10
1 5 9

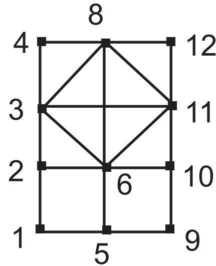
Choose as pivot 7

$$
\begin{pmatrix}
1 & * & & & * & & & & & & & \\
* & 2 & * & & & * & & & & & & \\
& * & \boxed{3} & * & & \boxed{} & \boxed{*} & \boxed{} & & \boxed{} & & \\
& & * & 4 & & & & * & & & & \\
* & & & & 5 & * & & & * & & & \\
& * & & & * & 6 & \boxed{*} & & & * & & \\
& & \boxed{*} & & & \boxed{*} & \boxed{7} & \boxed{*} & & \boxed{} & * & \\
& & & * & & & \boxed{*} & 8 & & & & * \\
& & & & * & & & & 9 & * & & \\
& & & & & * & & & * & 10 & * & \\
& & \boxed{} & & & \boxed{} & \boxed{*} & \boxed{} & & \boxed{*} & 11 & * \\
& & & & & & & * & & & * & 12
\end{pmatrix}
$$

leads to fill in

Fill in: pattern of row J is added to non zero entries in column 7
$\Longrightarrow$ indices connected with row 7 give a dense submatrix
here the submatrix to row/column 3, 6, 8 and 11 gets dense

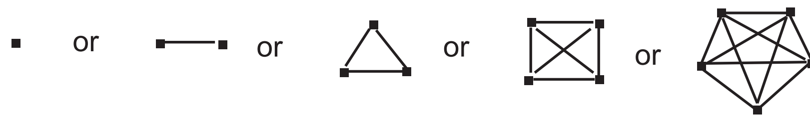
New graph:



one step GE in the graph consists in

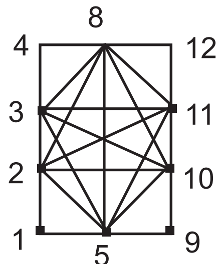
- remove edge 7
- add vertices such that all neighbours of 7 get fully connected

Definition: A fully connected graph is called 'clique', e.g.



In each elimination step the pivot knot is removed (pivot row and column are removed) and a subclique in the graph is generated.
Connecting all neighbours of the pivot entry.

Next step in GE: with pivot 6:   neighbours: 2, 3, 5, 8, 10, 11



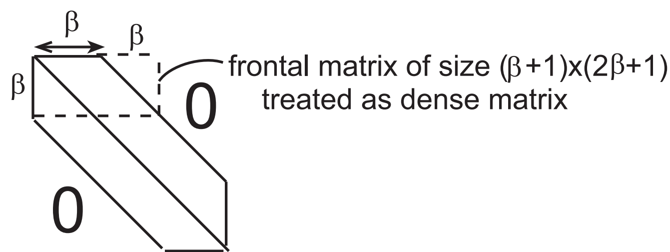
Gaussian elimination can be modelled without numerical computations only by computing the graphs algebraically.

Advantages:

- algebraic prestep is cheap
- gives information on the data structure (pattern) of resulting matrices
- shows whether Gaussian Elimination makes sense
- formulation in cliques, because in the course of GE, there will appear more and more cliques: cliques give short discretization of the graphs

4.5 Different direct solvers

Frontal methods for band matrices



- Apply first GE step with column pivoting in dense frontal matrix
- compute next row/column and move frontal matrix one step right+down

Multifrontal method for general sparse matrices

example: $A = \begin{pmatrix} * & 0 & * & * \\ 0 & * & * & * \\ * & * & * & 0 \\ * & * & 0 & * \end{pmatrix}$

d_{11} first pivot element is related to first frontal matrix, that contain all numbers related to one step GE with

a_{11} : $\frac{a_{i1}a_{1j}}{a_{11}}$: in dense submatrix:

a_{11}	a_{13}	a_{14}
a_{31}	$\frac{a_{31}a_{13}}{a_{11}}$	$\frac{a_{31}a_{14}}{a_{11}}$
a_{41}	$\frac{a_{41}a_{13}}{a_{11}}$	$\frac{a_{41}a_{14}}{a_{11}}$

Because $a_{12} = 0$, wee can in parallel consider a_{22} and the frontal matrix, related to the one step GE with a_{22}:

a_{22}	a_{23}	a_{24}
a_{32}	$\frac{a_{32}a_{23}}{a_{22}}$	$\frac{a_{32}a_{24}}{a_{22}}$
a_{42}	$\frac{a_{42}a_{23}}{a_{22}}$	$\frac{a_{42}a_{24}}{a_{22}}$

The computations $a_{ij} \to a_{ij} - \frac{a_{i1}a_{1j}}{a_{11}}$ and $a_{ij} \to a_{ij} - \frac{a_{i2}a_{2j}}{a_{22}}$ are independent and can be done in parallel.

5 Iterative methods for sparse matrices

X_0 initial guess (e.g $X_0 = 0$)
Iteration function ϕ : $x_{k+1} = \phi(x_k)$ gives sequence $|x_0, x_1, x_2, x_3, x_4, \dots|$
should converge $x_k \stackrel{k\to\infty}{\longrightarrow} \bar{x} = A^{-1}b$ (fast convergence)

Advantage: computation of $\phi(x)$ needs only matrix-vector products.
Do not change the pattern. It is easy to parallelize. Big question: fast convergence?

5.1 stationary methods

5.1.1 Richardson Iteration for Solving $Ax = b : \bar{x} := A^{-1}b$

$b = (A - I + I)x = (A - I)x + x \Longrightarrow x = b + (I - A)x = b + Nx$
Fix point iteration $x = \phi(x)$ with $\phi(x) = b + Nx$

x_0 start $x_{k+1} = \phi(x_k) = b + Nx_k$
if x_k convergent, $x_k \to \tilde{x}$, then $\tilde{x} = b + N\tilde{x} \Rightarrow A\tilde{x} = b \Rightarrow \hat{x} = \tilde{x}$
other formulation:
$\phi(x) = b + x - Ax = x + (b - Ax) = x + r(x)$ r - residual

Convergence analysis via "Neumann Series"

$$
\begin{aligned}
x_k &= b + Nx_{k-1} = b + N(b + Nx_{k-2}) = b + Nb + Nx_{k-2} \\
&= b + Nb + N^2b + Nx_{k-3} = \dots = b + Nb + \dots + N^{k-1}b + N^k x_0 \\
&= (\sum_{i=0}^{k-1} N^i)b + N^k x_0
\end{aligned}
$$

Special case: $x_0 = 0$: $x_k = (\sum_{j=0}^{k-1} N^j)b$

$\Rightarrow x_k \in span(b, Nb, N^2b, \dots, N^{k-1}b) = span(b, Ab, A^2b, \dots, A^{k-1}b) = K_k(A, b)$
$=$ Krylov-row of dimension k to matrix A and vector b, assume $\|N\| < 1$:

then $\sum_{j=0}^{k-1} N^j \stackrel{convergence}{\longrightarrow} \sum_{j=0}^{\infty} N^j = (I - N)^{-1} = A^{-1}$ $(\sum_{j=0}^{\infty} q^j = \frac{1}{1-q})$

$\Rightarrow x_k \to (\sum_{j=0}^{\infty} N^j)b = (I - N)^{-1}b = (I - (I - A))^{-1}b = A^{-1}b = \bar{x}$

Richardson gives convergent sequence if "A=I"
Error: $e_k := x_k - \hat{x}$

$e_{k+1} = x_{k+1} - \hat{x} = \underbrace{(b + Nx_k)}_{\phi(x_k)} - \underbrace{(b + N\hat{x})}_{\phi(\hat{x})} = N(x_k - \hat{x}) = Ne_k$

$\|e_k\| \leq \|N\|\|e_{k-1}\| \leq \|N\|^2\|e_{k-2}\| \leq \ldots \leq \|N\|^k\|e_0\|$

$\|N\| < 1 \Rightarrow \|N\|^k \overset{k\to\infty}{\longrightarrow} 0 \Rightarrow \|e_k\| \overset{k\to\infty}{\longrightarrow} 0 \quad ; \quad \rho(N) = \rho(I - A) < 1$

largest absolute value of an eigenvalue $< 1 \Rightarrow$ define a norm with $\|A\| < 1$
Eigenvalues of A have to be in a circle around 1 with radius 1.

5.1.2 Better splitting of A

$A := M - N$
Modifications of Richardson to get better convergence.
$b = Ax = (M - N)x = Mx - Nx \quad \Leftrightarrow \quad x = M^{-1}b + M^{-1}Nx$

$$\begin{aligned} new\ \phi(x) &= M^{-1}b + M^{-1}Nx = M^{-1}b + M^{-1}(M - A)x \\ &= M^{-1}(b - Ax) + x = x + M^{-1}r(x) \end{aligned}$$

M should be simple (easy to solve)
$x_{k+1} = M^{-1}b + M^{-1}Nx_k = x_k + M^{-1}(b - Ax_k) = x_k + M^{-1}r_k$

is equivalent to Richardson applied on $M^{-1}Ax = M^{-1}b$

Therefore convergent for $\rho(M^{-1}N) = \rho(I - M^{-1}A) < 1$
M is also called a precondition, because $M^{-1}A$ should be better conditioned than A itself: $M^{-1}A \approx I$

5.1.3 Jacobi (Diagonal) - Splitting:

$A = M - N = D - (L + U)$
with L: lower triangular, U: upper triangular, D: diagonal part of A

$x_{k+1} = D^{-1}b + D^{-1}(L + U)x_k = D^{-1}b + D^{-1}(D - A)x_k = x_k + D^{-1}r_k$
convergent if $\rho(M^{-1}N) = \rho(I - D^{-1}A) < 1$

elementwise:

$$x_j^{k+1} = \frac{1}{a_{jj}}(b_j - \sum_{\substack{m=1 \\ m\neq j}}^{n} a_{jm}x_m^{(k)})$$

or

$$a_{jj}x_j^{k+1} = b_j - \sum_{m=1}^{j-1} a_{jm}x_m^{(k)} - \sum_{m=j+1}^{n} a_{jm}x_m^{(k)}$$

To improve convergence: $x_{k+1} = x_k + D^{-1}r_k$ $\quad D^{-1}r_k-$ correction step

$x_{k+1} = x_k + \overbrace{\omega}^{include\ damping} D^{-1}r_k$ with step length ω: damped Jacobi

$$\begin{aligned} x_{k+1} &= x_k + \omega D^{-1}r_k = x_k + \omega D^{-1}(b - Ax_k) = x_k + \omega D^{-1}b - \omega D^{-1}Ax_k \\ &= (I - \omega D^{-1}A)x_k + \omega D^{-1}b = (I - \omega D^{-1}(D - L - U))x_k + \omega D^{-1}b \\ &= \omega D^{-1}b + [(1-\omega)I + \omega D^{-1}(L+U)]x_k \end{aligned}$$

convergent if $\rho([\underbrace{(1-\omega)I + \omega D^{-1}(L+U)}_{\text{I for } \omega\to 0}]) < 1$

Jacobi method is easy to parallelize: for $\omega = 1$: Jacobi: look for optimal ω
only A * vector, D^{-1} * vector
To improve convergence (Block Jacobi):

A =
-U
-L
D

5.1.4 Gauss-Seidel method by improving convergence

$$a_{jj}x_j^{(k+1)} = b_j - \sum_{m=1}^{j-1} a_{jm}x_m^{(k+1)} - \sum_{m=j+1}^{n} a_{jm}x_m^{(k)} \qquad j = 1, 2, ..., n$$

Try to use the newest available information in each step.
Advantage: fast convergence

$$a_{kk}x_j^{(k+1)} = b_j - \sum_{m=1}^{j-1} a_{jm}x_m^{(k+1)} - \sum_{m=j+1}^{n} a_{jm}x_m^{(k)}$$

$Dx_{k+1} = b + Lx_{k+1} + Ux_k \quad$ or $\quad (D-L)x_{k+1} = b + Ux_k$
is related to splitting $A = \underbrace{(D-L)}_{M} - \underbrace{U}_{N}$: Gauss-Seidel-method
In each step we have to solve a triangular linear system:
disadvantage in parallel!
Data depending graphs for iteration methods reorder A colouring of the graph red-block (circle):

$$\begin{matrix} \circ & \times & \circ & \times \\ \times & \circ & \times & \circ \\ \circ & \times & \circ & \times \\ \times & \circ & \times & \circ \end{matrix} \Longrightarrow \begin{pmatrix} \times \\ \times \\ \vdots \\ \times \\ \hline \circ \\ \circ \\ \vdots \\ \circ \end{pmatrix}$$ compromise: convergence $\leftrightarrow$ parallelism

convergence depending on $\rho(1-(D-L)^{-1}A) < 1$

$(D-L)^{-1}Ax = (D-L)^{-1}b$

Damping: $\quad x_{k+1} = x_k + \omega(D-L)^{-1}r_k$
Stationary methods can be written in the form in general:
$x_{k+1} = C + Bx_k$ with constant vector C and iteration matrix B

$$= x_k + \underbrace{F}_{preconditioner} r_k$$

$\rho(B) < 1$ convergence $\quad \underline{B = I - FA}$

5.2 Nonstationary Methods

5.2.1 Let A symmetric positive definite $A = A^T > 0$ (spd.)

Consider function $\phi(x) = \frac{1}{2}(x^TAx) - b^Tx$

Derivative: $\quad \nabla\phi(x) = Ax - b$ gradient ϕ Paraboloid
$\bar{x} \quad$ Minimum of ϕ is unique with $\quad \nabla\phi(\bar{x}) = A\bar{x} - b = 0 \Leftrightarrow A\bar{x} = b$
Compute $\bar{x}$, solution of $Ax = b$ by approximating minimum of ϕ iteratively

x_k last iterate: find $x_{k+1} = x_k + \lambda v$ with search direction v and stepsize λ, such that $\phi(x_{k+1}) < \phi(x_k)$ and hence x_{k+1} is nearer to minimum.
Search direction v:

$\frac{d}{d\lambda}\phi\left(x_k + \lambda v\right)\big|_{\lambda=0} = \nabla\phi(x_k)v$ (directional derivative) $\overset{!}{<} 0$

Optimal search direction $\underline{v = -\nabla\phi(x_k)} = b - Ax_k = r_K \rightarrow \underline{x_{k+1} = x_k + \lambda r_k}$

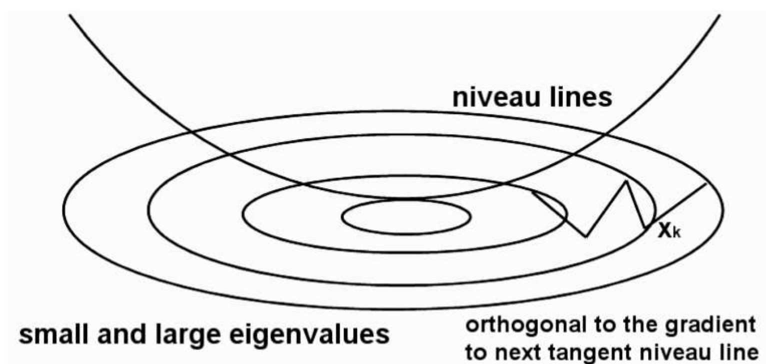
stepsize λ : finding $\min\limits_{\lambda} \phi(x_k + \lambda r_K)$ is a simple 1D-problem

$$\begin{aligned}
\frac{d}{d\lambda}\phi(x_k + \lambda r_k) &= \frac{d}{d\lambda}\left(\frac{1}{2}(x_k^T + \lambda r_k^T)A(x_k + \lambda r_k) - b^T(x_k + \lambda r_k)\right) \\
&= \frac{d}{d\lambda}\left(\frac{1}{2}x_k^T A x_k + \lambda r_k^T A x_k + \frac{\lambda^2}{2} r_k^T A r_k - b^T x_k - \lambda b^T r_k\right) \\
&= r_k^T A x_k - b^T r_k + \lambda r_k^T A r_k \\
&= -r_k^T r_k + \lambda r_k^T A r_k \overset{!}{=} 0
\end{aligned}$$

$\Leftrightarrow \lambda = \frac{r_k^T r_k}{r_k^T A r_k} \qquad v_k = r_k$

Algorithm: $x_{k+1} = x_k + \frac{r_k^T r_k}{r_k^T A r_k} r_k$ with $r_k = b - Ax_k$

Gradient Method, steepest decent.
locally optimal search directions are not globally optimal, if paraboloid is very distorted $\Leftrightarrow$ very small and large eigenvalues.
if condition(A) $= \|A\|_2\|A^{-1}\|_2 = \frac{\lambda_{max}}{\lambda_{min}} >> 1 \Leftrightarrow cond(A) >> 1$
guaranteed convergence, but mostly slow!



To analyse this slow convergence also theoretically, we introduce the following norm (so-called A-norm)

$\|x\|_A := \sqrt{x^T A x}$

Then it holds for the error $x - \bar{x}$ with $\bar{x} = A^{-1}b$

$$\|x-\bar{x}\|_A^2 = \|x-A^{-1}b\|_A^2 = (x-A^{-1}b)^T A(x-A^{-1}b) =$$
$$= x^T Ax - 2b^T x + b^T A^{-1} b = 2\phi(x) + b^T A^{-1} b$$

Hence minimizing ϕ is equivalent to minimizing the error in the A-norm.

with $\quad x_{j+1} := x_j + \lambda_j r_j, \quad r_j = b - Ax_j \ and \ \lambda_j = \frac{r_j^T r_j}{r_j^T A r_j}$

we get the following inequality between $\phi(x_{j+1})$ and $\phi(x_j)$:

$$\begin{aligned}
\phi(x_{j+1}) &= \tfrac{1}{2}(x_j^T + \lambda_j r_j^T) A (x_j + \lambda_j r_j) - (x_j^T + \lambda_j r_j^T) b \\
&= \tfrac{1}{2} x_j^T A x_j + \lambda_j x_j^T A r_j + \tfrac{\lambda_j^2}{2} r_j^T A r_j - x_j^T b - \lambda_j r_j^T b \\
&= \phi(x_j) + \lambda_j r_j^T (Ax_j - b) + \tfrac{\lambda_j^2}{2} r_j^T A r_j \\
&= \phi(x_j) - \tfrac{r_j^T r_j}{r_j^T A r_j} r_j^T r_j + \tfrac{1}{2} \tfrac{(r_j^T r_j)^2}{(r_j^T A r_j)^2} r_j^T A r_j \\
&= \phi(x_j) - \tfrac{1}{2} \tfrac{(r_j^T r_j)^2}{(r_j^T A r_j)} \\
&= \phi(x_j) - \tfrac{1}{2} \underbrace{\tfrac{(r_j^T r_j)^2}{r_j^T A r_j r_j^T A^{-1} r_j}}_{\rho_j} r_j^T A^{-1} r_j \\
&= \phi(x_j) - \tfrac{1}{2} \rho_j (b - Ax_j)^T A^{-1} (b - Ax_j) \\
&= \phi(x_j) - \tfrac{\rho_j}{2} (b^T A^{-1} b + x_j^T A x_j - 2b^T x_j) \\
&= \phi(x_j) - \rho_j (\phi(x_j) + \tfrac{1}{2} b^T A^{-1} b)
\end{aligned}$$

$$\Rightarrow \phi(x_{j+1}) + \tfrac{1}{2} b^T A^{-1} b = \phi(x_j) + \tfrac{1}{2} b^T A^{-1} b - \rho_j (\phi(x_j) + \tfrac{1}{2} b^T A^{-1} b)$$

$\Rightarrow \underline{\|x_{j+1} - \bar{x}\|_A^2 = \|x_j - \bar{x}\|_A^2 \ (1 - \rho_j)}$ error in next step

$$\rho_j = \frac{(r_j^T r_j)^2}{r_j^T A r_j r_j^T A^{-1} r_j} \geq \frac{1}{\lambda_{max} \underbrace{1/\lambda_{min}}_{\lambda_{max}(A^{-1})}} = \frac{1}{cond(A)}$$

$(range(A) = \{\frac{r_j^T A r_j}{r_j^T r_j} | r_j \in A\} \quad [\lambda_{min}(A), \lambda_{max}(A)])$

$\Rightarrow \|x_{j+1} - \bar{x}\|_A^2 = (1 - \frac{1}{cond(A)}) \|x_j - \bar{x}\|_A^2$

Is therefore $cond(A) >> 1$, then the improvement in every iteration step is nearly nothing.

Therefore we have very slow convergence!

5.2.2 Improving the gradient method $\to$ conjugate gradients

Ansatz: $\quad x_{k+1} = x_k + \alpha_k p_k \quad (\alpha_k$ stepsize $\quad ; \quad p_k$ search direction)

As search direction we do not use the gradient, but a modification of the gradient.
Choose new search direction such that p_k is orthogonal to $p_j \Leftrightarrow p_k^T A p_j = 0$

We choose the new search direction as the projection of the gradient on the A-conjugate subspace relative to previous p_k.

α_k derived by I-dimensional minimization as before

Algorithm: $\qquad x_0 = 0, \quad r_0 = b - Ax_0$
for $k = 1, 2, \ldots :\quad \beta_{k-1} = r_{k-1}^T r_{k-1} / r_{k-2}^T r_{k-2} \qquad \beta_0 = 0$

$$p_k = r_{k-1} + \beta_{k-1} p_{k-1} \qquad (p_k \text{ A-conjugate to } p_{k-1}, p_{k-2}, \ldots)$$

$$\alpha_k = r_{k-1}^T r_{k-1} / p_k^T A p_k \qquad \text{(1-dimension min.)}$$

$$x_k = x_{k-1} + \alpha_k p_k$$

$$r_k = r_{k-1} - \alpha_k A p_k$$

if $\|r_k\| < \epsilon$: stop $\quad$ Conjugate gradient method

Main properties of the computed vectors:

$$p_j^T A p_k = 0 = r_j^T r_k \qquad \text{for } j \neq k$$

$$span(p_1, \ldots, p_j) = span(r_0, \ldots, r_{j-1}) = span(r_0, Ar_0, \ldots, , A^{(j-1)} r_0) = K_j(A, r_0)$$

(Krylov subspaces)

especially for $x_0 = 0$: $\qquad span(b, Ab, \ldots, A^{(j-1)} b) = K_j(A, b)$

x_k is the best approximate solution in subspace $K_k(A, b)$

for $x_0 = 0$: $\ x_k \in span(b, Ab, \ldots, A^{(j-1)} b)$ and $\|x_k - \bar{x}\|_A = \min_{x \in K(A,b)} \|x - \bar{x}\|_A$

$\to$ main property: $\bar{x} = A^{-1} b$

Choosing these special search directions, the 1D minimization gives us the best solution relative to a k-dimensional subspace. In each step optimal solution to larger and larger subspaces!

Consequence: after n steps: $\quad K_n(A, b) = \mathbf{R^n}$

$\Longrightarrow x_n = \bar{x} \quad$ in exact arithmetic or: $\quad \min_{x_k \in K_n} \|x_k - \bar{x}\|_A = 0$

Unfortunately, this is only true in exact arithmetic.
Also convergence after n steps would be not good enough.

error estimation for $x_0 = 0$:

$$\|e_k\|_A = \|x_k - \bar{x}\|_k = \min_{x \in K_k(a,b)} \|x - \bar{x}\|_A = \min_{\alpha_j} \| \sum_{j=0}^{k-1} \alpha_j (A^j b) - \bar{x}\|_A$$

$$= \min_{p_{k-1}(x)} \|p_{k-1}(A)b - \bar{x}\|_A = \min_{p_{k-1}(x)} \|p_{k-1}(A)A\bar{x} - \bar{x}\|_A$$

$$= \min_{q_k(0)=1} \|q_k(A) \underbrace{(\bar{x} - x_0)}_{e_0} \|_A = \|e_k\|_A$$

a spd. $\Rightarrow A = U \Lambda U^T$ (Λ: diagonal matrix/eigenvalues, in U: eigenvectors)

we can write: $e_0 = \sum_{j=1}^{n} \xi_j u_j$ $\quad$ ($u_1, ..., u_n$ are ONB of eigenvectors of A)

$$\Rightarrow \|e_k\|_A = \min_{q_k(0)=1} \|q_k(A) \overbrace{\sum_{j=1}^{n} \xi_j u_j}^{e_0} \|_A$$

$$= \min_{q_k(0)=1} \| \sum_{j=1}^{n} \xi_j q_k(A) u_j \|_A$$

$$= \min_{q_k(0)=1} \| \sum_{j=1}^{n} \xi_j q_k(\lambda_j) u_j \|_A$$

$$\leq \min_{q_k(0)=1} [\max_{j=1}^{n} |q_k(\lambda_j)| \| \sum_{j=1}^{n} \xi_j u_j \|]$$

$$= \min_{q_k(0)=1} [\max_{j=1}^{n} |q_k(\lambda_j)|] \|e_0\|_A$$

by choosing any polynomial with $q_k(0) = 1$ and degree k, we can derive estimates for the error e_k

e.g.: $q_k(x) := (1 - \frac{2}{\lambda_{max} + \lambda_{min}} x)^K$ leads to:

$$\begin{aligned}
\|e_k\|_A &\leq \max_{j=1}^{n} |q_k(\lambda_j)| \|e_0\|_A = \max_{j=1}^{n} |1 - \frac{2}{\lambda_{max} + \lambda_{min}} \lambda_j|^K \|e_0\| \\
&= |(1 - \frac{2\lambda_{max}}{\lambda_{max} + \lambda_{min}})^k \|e_0\|_A \\
&= (\frac{\lambda_{min} - \lambda_{max}}{\lambda_{max} + \lambda_{min}})^k \|e_0\|_A \\
&= (\frac{cond(A) - 1}{cond(A) + 1})^k \|e_0\|_A
\end{aligned}$$

Better estimates by normalized Chebychev polynomials: $T_n(x) = cos(n \, arccos(x))$

$$\|e_k\|_A \leq \frac{1}{T_k(\frac{cond(A)+1}{cond(A)-1})} \leq 2 \left(\frac{\sqrt{cond(A)} - 1}{\sqrt{cond(A)} + 1} \right)^k$$

e.g. assume that A has only two eigenvalues λ_1 and λ_2

set $q_2(x) := \frac{(\lambda_1 - x)(\lambda_2 - x)}{\lambda_1 \lambda_2} \Rightarrow \quad q_2(0) = 1 \Rightarrow \quad \|e_2\|_A \leq \max\limits_{j=1,2} |q_2(\lambda_j)| \|e_0\|_A = 0$

$\Rightarrow$ convergence of cg-method after 2 steps!

Similar behaviour for eigenvalue clusters. After 2 steps: small error.

5.2.3 GMRES for General Matrix A, not spd

Consider small subspaces U_m and determine optimal approximate solutions in these subspaces for $Ax = b$ in U_m.

so we restrict x to the form $x = U_m y$: $\min\limits_{x \in U_m} \|Ax - b\|_2 = \min\limits_y \|A(U_m y) - b\|_2$

could be solved by normal equations $\underline{U_m^T A^T A U_m y = U_m^T A^T b}$

What subspace U_m should we choose? (relative to Ax=b):

$U_m := U_m(A, b) = span(b,\ Ab, \ldots, A^{m-1}b)$ (bad basis for U_m)

First step: provide Orthonormal basis for $U_m(A, b)$:

$u_1 := b / \|b\|_2$

for $j = 2 : m$

$$\tilde{u}_j := Au_{j-1} - \sum_{k=1}^{j-1} \underbrace{(u_k^T A u_{j-1})}_{h_{k,j-1}} u_k \quad \Rightarrow \tilde{u}_j \perp u_1, ..., u_{j-1}$$

$$u_j := \tilde{u}_j / \underbrace{\|\tilde{u}_j\|_2}_{h_{j,j-1}}$$

end.

$$Au_{j-1} = \sum_{k=1}^{j-1} (u_k^T A u_{j-1}) u_k + \tilde{u}_j = \sum_{k=1}^{j-1} h_{k,j-1} u_k + h_{j,j-1} u_j = \sum_{k=1}^{j} h_{k,j-1} u_k$$

$$AU_m = A(u_1, \ldots, u_m) = (u_1, \ldots, u_{m+1}) \tilde{H}_{m+1,m} = \tilde{U}_m \tilde{H}_{m+1,m}$$

$$\text{with } \tilde{H}_{m+1,m} = \begin{pmatrix} h_{11} & \ldots & \ldots & h_{1m} \\ h_{21} & \ddots & & \vdots \\ 0 & \ddots & \ddots & \\ \vdots & \ddots & \ddots & h_{m,m} \\ 0 & \ldots & 0 & h_{m+1,m} \end{pmatrix} \text{ Upper m+1×m Hessenberg form}$$

Now we can solve the minimization problem:

$$\begin{aligned}
\min_{x\in U_m} \|Ax-b\|_2 &= \min_y \|A(U_m y)-b\|_2 \\
&= \min_y \|U_m\tilde{H}_{(m+1,m)}y - \|b\|u_1\|_2 \\
&= \min_y \|U_m(\tilde{H}_{(m+1,m)}y - \|b\|e_1)\|_2 \\
&= \min_y \|\tilde{H}_{(m+1,m)}y - \|b\|e_1\|_2
\end{aligned}$$

because U_m is part of an orthogonal matrix. (invariant)

We can use Givens rotation to compute a QR-decomposition of the upper Hessenberg matrix $\tilde{H}_{(m+1,m)}$

$$G_1\begin{pmatrix} * & & & * \\ & \ddots & & \\ & \ddots & & \\ & & * & * \end{pmatrix},\quad G_2\begin{pmatrix} * & * & & & * \\ 0 & * & & & \\ & * & \ddots & & \\ & & & * & * \\ & & & & * \end{pmatrix},\dots,G_m\begin{pmatrix} * & & & & \\ 0 & * & & & \\ & & \ddots & & \\ & & & & * \\ & & & 0 & * \end{pmatrix}$$

gives $Q\tilde{H}_{(m+1m)} = G_m\dots G_2G_1\tilde{H}_{(m+1m)} = R = \begin{pmatrix} \cdots\cdots \\ \ddots\ \vdots \\ 0\dots 0 \end{pmatrix} = \begin{pmatrix} R_m \\ 0 \end{pmatrix}$

$$\begin{aligned}
\min_{x\in U_m}\|Ax-b\|_2 &= \min_y \|\tilde{H}_{(m+1,m)}y - \|b\|e_1\|_2 \\
&= \min_y \|\begin{pmatrix} R_m y \\ 0 \end{pmatrix} y - \underbrace{\|b\|G_m\dots G_1}_{\tilde{b}_m} e_1\|_2 \\
&= \min_y \left\|\begin{pmatrix} R_m y \\ 0 \end{pmatrix} - \tilde{b}_m\right\|_2 \\
&= \min_y \left\|\begin{pmatrix} R_m y - \tilde{b}_1 \\ -\tilde{b}_2 \end{pmatrix}\right\|_2
\end{aligned}$$

Solution: $\quad R_m y = \tilde{b}_1 \to Y$

<u>GMRES:</u>

- Compute $\tilde{H}_{(m+1,m)}$ by Arnoldi-orthogonalization
- compute QR-factorization
- solve least squares problem $\longrightarrow x_k$

Iterative: enlarge U_m to $A^m U_m \longrightarrow$ new column in $\tilde{H}_{(m+1,m)}$
$\longrightarrow$ new Givens matrix update QR $\longrightarrow$ new column in R
$\longrightarrow$ solve enlarged LS by updating x_k

Gets very costly after 50 steps. Restarted version: GMRES(20)$\rightarrow \tilde{x}$

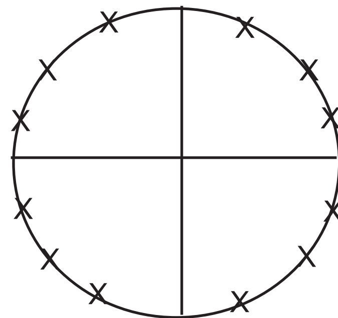
$A(\tilde{x} - x) = b \Rightarrow Ax = b - A\tilde{x} = \tilde{r}$. Call GMRES(20) for $Ax = \tilde{r}$

$$\begin{aligned}
\|r_m\|_2 &:= \|Ax_m - b\|_2 = \min_{x \in U_m} \|Ax - b\|_2 = \min_{x_j} \|A[\sum_{j=0}^{m-1} \alpha_j (A^j b)] - b\|_2 \\
&= \min_{p_{m-1}} \|Ap_{m-1}(A)b - b\|_2 = \min_{q_m(0)=1} \|q_m(A)b\|_2 = \min_{q_m(0)=1} \|Vq_m(1)V^{-1}b\|_2 \\
&\leq \|V\|_2 \|V^{-1}\|_2 \underbrace{\|b\|_2}_{\|r_j\|_2} [\max_{j=1}^{n} \min_{q_m(0)=1} \|q_m(\lambda_j)\|] = cond|V| \underbrace{\|r_j\|_2 \min_{q_m(0)=1} \max_{j=1}^{n} |q_m(\lambda_j)|}_{like\ cg.}
\end{aligned}$$

5.2.4 Convergence of cg. or GMRES

Convergence of cg./GMRES depends strongly on the position of eigenvalues.

$$A = \begin{pmatrix} 0 & & & 1 \\ 1 & \ddots & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{pmatrix}$$



GMRES needs n steps!

<u>Preconditioning</u>: Improve the eigenvalue location of A:

replace the given $Ax = b$ by
$$P^{-1}Ax = P^{-1}b \ (implicit) \ (P \approx A)$$
or
$$MAx = Mb \ (explicit) \ (M \approx A^{-1})$$

$(P_1^{-1} A P_2^{-1})(P_2 x) = P_1^{-1} b \Leftrightarrow \tilde{A}\tilde{x} = \tilde{b}$

symmetric: $(P_1^{-1}AP_1^{-T})(P_1^T x) = (P_1 b)$ ($\tilde{A}$ should have clustered eigenvalues)

stationary methods: $\quad A = M - N \quad ; \quad b = Ax = (M-N)x = Mx - Nx$

$\Rightarrow x_{k+1} = M^{-1}b + M^{-1}Nx_k = M^{-1}b + (I - M^{-1}A)x_k$

convergent iff $\|I - M^{-1}A\| < 1 \Leftrightarrow$ eigenvalues of $M^{-1}A$ are clustered near 1

Good splitting $\rightarrow$ good precondition

Improve stationary methods by using the related splitting as preconditions in cg. or GMRES:

(i) Jacobi-splitting with $D = diag(A) \longrightarrow$ Jacobi preconditioner $M := D$

(ii) Gauss-Seidel splitting $M := L + D$

(iii) ILU = incomplete LU decomposition:

Apply GE-algorithm, but reduced cg. on the pattern of the sparse matrix A.

$$A = \begin{pmatrix} * & & & * \\ & * & & \\ & & * & \\ * & & & * \end{pmatrix} \rightarrow \text{to } L = \begin{pmatrix} * & & & 0 \\ & * & & \\ & & * & \\ * & & & * \end{pmatrix},\ U = \begin{pmatrix} * & & & * \\ & * & & \\ & & * & \\ 0 & & & * \end{pmatrix}$$

$A = LU + R$

<u>Modification:</u>
ILU(0) related pattern of A ; ILU(1) related to L(0)U(0)

ILUT: Treshhold ILU: Apply standard GE, but in each step sparsification by deleting all entries less or equal $\sum$.

MILU: Modified ILU:

$$\begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}$$

Apply GE with sparsification. Move all deleted entries to the diagonal.

IC (incomplete cholesky - symmetric form of ILU) implicite preconditioners.

disadvantage: in each step we have to solve sparse triangular system
ILU $\rightarrow$ L,U hard to parallelize
<u>Idea:</u> explicit preconditioner $M \approx A^{-1}$

to minimize $\|AM - I\|_?$ <u>choose Frobenius norm</u> $\|B\|_F^2 = \sum_{j=1}^{n} (B_{.j})_2^2 = \sum_{i=1}^{n} (B_{i.})_2^2$

choose matrix class polynomial preconditioner:

$A^{-1}(A^n+\gamma_{n-1}A^{n-1}+\ldots+\gamma_1 A+\gamma_0) \equiv 0 \quad \Rightarrow \quad \gamma_0 A^{-1} = -A^{n-1}-\gamma_{n-1}A^{n-2}-\ldots-\gamma_1$

$min\|I - p_m(A)A\|$: $\underline{\min_{p_m \in P_m} \max_{\lambda_1,\ldots\lambda_m} |1 - p_m(\lambda)\lambda|}$

Assume that the eigenvalues in interval: $\underline{0 < c \leq \lambda \leq d < \infty}$

for $\min_{p_m \in P_m} \max_{\lambda \in [c,d]} |1 - p_m(\lambda)\lambda| \quad \Rightarrow \quad$ solution: $p_m(x) = \frac{T_{m+1}\left(\frac{d+c}{d-c}\right) - T_{m+1}\left(\frac{d+c-2x}{d-c}\right)}{xT\left(\frac{d+c}{d-c}\right)}$

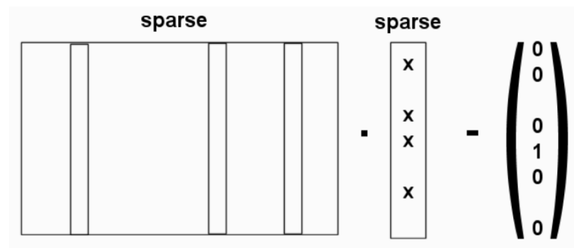
(transformation from oscillation $[-1, 1]$ to $[c, d]$)

cg., GMRES are optimal in Krylov spaces $(b, Ab, A^2b, \ldots)$
$p_m(A)b$ is easy to parallelize, but not optimal.

choose M: sparse matrices, same sparcity as A

$$\min_{M \in P(A)} \|AM - I\|_F^2 = \min_{M \in P(A)} \sum_{j=1}^{n} \|(AM - I)e_j\|_2^2 = \sum_{j=1}^{n} \min_{M \in P(A)} \| \underbrace{AM_j - e_j}_{vectors} \|_2^2$$

n independent minimization problems for computing $M_1, M_2, \ldots, M_n$



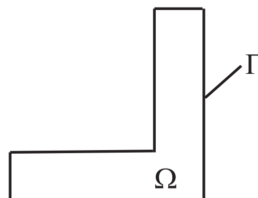
$\Leftrightarrow A(\,:, I_j)M(I_j) - e_j$ (I_j are the non-zero entry indices of M_j)

$J_j :=$ indices of non-zero rows of $A(\,:, I_j) \rightarrow \underline{min\|A(J_j, I_j)M(I_j) - e_j(J_j)\|}$

Least squares problem can be solved by QR-method, Givens or Householder. Solve n independent small LS problems, to get M. To apply this preconditioner in cg. or GMRES: we only have to multiply sparse M times vector.

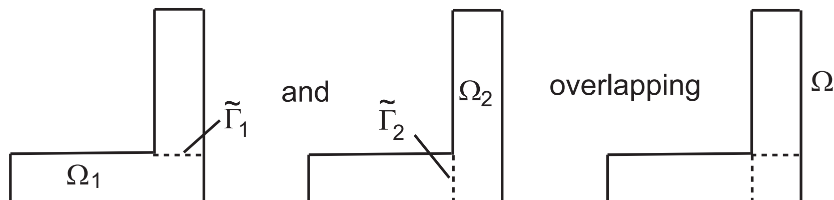
6 Collection remaining problems

6.1 Domain Decomposition Methods for Solving PDE



region Ω with boundary Γ
Given PDE, e.g. Laplace equation: $\Delta u = u_{xx} + u_{yy} = \frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2} \stackrel{!}{=} f(x,y)$
in Ω and $u|_\Gamma = q$ Dirichlet problem

How to parallelize?



Γ_1 boundary of Ω_1 with $\tilde{\Gamma}_1$ unknown values

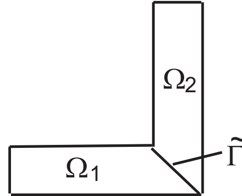
Γ_2 boundary of Ω_2 with $\tilde{\Gamma}_2$ unknown values

<u>Idea:</u>

<table>
<tr><td>Solve PDE Ω_1 with some
estimated boundary values for $\tilde{\Gamma}_1$</td><td>Solve PDE Ω_2 with some
some estimated boundary values for $\tilde{\Gamma}_2$</td></tr>
<tr><td colspan="2">Exchange boundary values on $\tilde{\Gamma}_1$ and $\tilde{\Gamma}_2$</td></tr>
</table>

overlapping Domain Decomposition

Second approach: Nonoverlapping DD $\leftrightarrow$ Dissection method:



$$\begin{pmatrix} A_1 & 0 & F_1 \\ 0 & A_2 & F_2 \\ G_1 & G_2 & A_3 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ \hat{u}_2 \\ \hat{u}_3 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix}$$

Solve by Schur complement or preconditioner ($S = A_3 - G_1 A_1^{-1} F_1 - G_2 A_2^{-1} F_2$):

$$M = \begin{pmatrix} A_1^{-1} & * & * \\ * & A_2^{-1} & * \\ * & * & 1 \end{pmatrix} \sim \text{ to solving PDE in } \Omega_1 \text{ and } \Omega_2$$

6.2 Parallel Computation of the Discrete Fourier Transformation

Definition: $\omega_n = e^{\frac{-2\pi i}{n}}$; $y = DFT(x)$; $y_k = \sum_{j=0}^{n-1} \omega_n^{kj} x_j \quad (k = 0, 1, ..., n-1)$

$$Y = \begin{pmatrix} 1 & 1 & \dots & 1 \\ 1 & \omega_n & * & \omega_n^{(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} x_0 \\ \vdots \\ \vdots \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} f_1 x \\ f_2 x \\ \vdots \\ f_n x \end{pmatrix}$$

collection of n independent dot-products
For a dot-product we can use fan-in algorithm: n-processors $\rightarrow O(logn)$ steps
In total: n $\times$ n processors $\rightarrow O(logn)$ steps for DFT
Complexity in parallel for DFT is $O(logn)$
Sequentially the complexity of the DFT is $O(n\ logn)$ (FFT-method)

What is a good sparsity pattern for M?

$$\left(\begin{array}{c|c} \text{A} & \text{B} \\ \hline \text{B} & \text{A} \end{array}\right)^{-1} = \begin{pmatrix} * & * \\ * & * \end{pmatrix}$$

A priori patterns for M:
$A, A^2, A^3, ...$ triangular ; $A^T, A^{2^T}, A^{3^T}, ...$ orthogonal, $(A^T A), (A^T A)^2$
Sparsification: Delete small entries in A, $(\epsilon), A_\epsilon, A_\epsilon^2, A_\epsilon^T, ...$

A priori static pattern for computing M.
Dynamic minimization, that finds a good pattern automatically:
We start with the diagonal pattern or with A_ϵ for example
$\rightarrow$ solve n related LS-problems $\rightarrow M_j$
How to find a better pattern?

$$\min_{\mu_k} \|A(M_j + \mu_k e_k) - e_j\|_2^2 = \min_{\mu_k} \| \underbrace{(AM_j - e_j)}_{r_j} + A\mu_k e_k\|_2^2$$

$\frac{d}{d\mu}\|r_j + A\mu_k e_k\|_2^2 \Rightarrow \mu_k = \frac{-r_j^T A e_k}{\|Ae_k\|_2^2}$ (most cases: $\mu_k = 0$)

improvement: $\|r_j + \mu_k A_j\|^2 = \|r_j\|_2^2 - \boxed{\frac{(r_j^T A e_k)^2}{\|Ae_k\|_2^2}}$

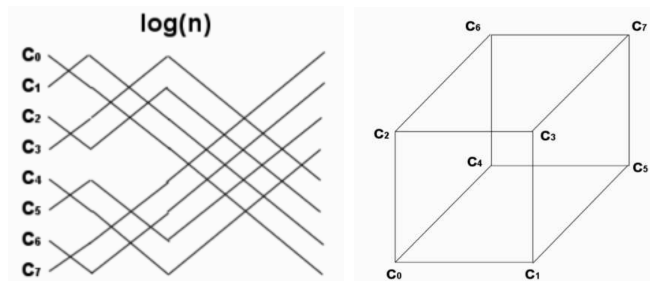
Factorized Sparse Approximation, Inverses: spd $A = A^T > 0$
$A = L_A^T L_A$ (Cholesky factorization) ; $A^{-1} \approx LL^T$

$min\|L_A L - I\|_F$ (sparsity structure of L) ; $min\|L_A(I,J)L_K(J) - e_K(I)\|$

<u>normal equations:</u>
$L_A^T(I,J)L_A(I,J)L_K(J) = L_A^T(I,J)e_K(I)$
$A(J,J)L_K(J) = \underbrace{L_{A,KK}}_{=1} e_K(I)$ (diagonal scaling)

First we compute L under the assumption $L_{A,KK} = 1$ (diagonal entries)
$D = L^T A L$; Replace L by $LD^{1/2}$

function $\overbrace{(v_0, \ldots, v_{n-1})}^{result} = IDFT\,\overbrace{(c_0, \ldots, c_{n-1}, n)}^{input}$
if $n == 1$ then $v_0 = c_0$; else $m = \frac{n}{2}$;
$z1 = IDFT(c_0, c_2, c_4, \ldots, c_{n-2}, m)$
$z2 = IDFT(c_1, c_3, c_5, \ldots, c_{n-1}, m)$
$for\ j = 0, \ldots, m-1$ $\qquad (\omega = e^{\frac{2\pi i}{n}})$
$v_j = z1 + \omega^j z2j$;
$v_{m+j} = z1 - \omega^j z2j$;
$end\ for$
end if.



6.3 Parallel Computation of Eigenvalues

$Ax = \lambda x \quad , \quad \lambda$ eigenvalue, $x \neq 0$ eigenvector

$A = A^T$ symmetric

$\underline{A = U\Lambda U^T}$ or $\underline{AU = U\Lambda}$ $(Au_j = \lambda u_j) \quad ; \quad UU^T = U^T U = I$

Standard: QR-Algorithm: $A_k = Q_k R_k \Rightarrow A_{k+1} = R_k Q_k = (Q_k^T A_k) Q_k$
(A_k: diagonal matrix)

Jacobi method:

$\|A\|_F^2 = \sum_{i,j=1}^{n} a_{ij}^2 \quad ; \quad \mathit{off}^{\,2}(A) = \|A\|_F^2 - \sum_{i=1}^{n} a_{ii}^2$

Looking for iterative method, $\mathit{off}\,(a_n) \to 0$
Givens or Jacobi rotations depending on p, q, θ

$$J(p,q,\theta) = \begin{pmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & c & & s & & & \\ & & & & \ddots & & & & \\ & & & -s & & c & & & \\ & & & & & & 1 & & \\ & & & & & & & \ddots & \\ & & & & & & & & 1 \end{pmatrix} \qquad cos(\theta) = c\ ,\ sin(\theta) = s$$

$A_k \longrightarrow J^T A_k J = A_{k+1} = B$

$$\begin{pmatrix} b_{pp} & b_{pq} \\ b_{qp} & b_{qq} \end{pmatrix} = \begin{pmatrix} c & s \\ -s & c \end{pmatrix}^T \begin{pmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{pmatrix} \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \stackrel{!}{=} \ Diagonal$$

$a_{pq}(c^2 - s^2) + (a_{pp} - a_{qq})cs \stackrel{!}{=} 0 \Rightarrow \theta \to cos, sin$
$a_{pp}^2 + 2a_{pq} + a_{qq}^2 = b_{pp}^2 + b_{qq}^2$

$$\begin{aligned} \mathit{off}^{\,2}(B) &= \|B\|_F^2 - \sum_{i=1}^{n} b_{ii}^2 \\ &= \|A\|_F^2 - \sum_{i=1; i\neq p,q}^{n} a_{ii}^2 - (a_{pp}^2 + 2a_{pq} + a_{qq}^2) = \mathit{off}^{\,2}(A) - 2a_{pq} \end{aligned}$$

Non diagonal entries are getting smaller. (largest effect by largest entries)
$(\theta_1, \theta_2) \Rightarrow (J_1^T A, J_2^T A) \Rightarrow \tilde{A} \Rightarrow (\tilde{A} J_1, \tilde{A} J_2) \Rightarrow A_{k+1}$