Parallel Scientific Computing

Lecture 4

Direct methods for linear systems

Performance analysis of parallel programs

Solution procedures for linear systems — Recap

Find $\mathbf{x} \in \mathbb{R}^n$ such that $\boxed{\mathbf{A}\mathbf{x} = \mathbf{b}}$ with $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\mathbf{b} \in \mathbb{R}^n$.

Solution procedures

- **Direct methods:** Factorization of $\mathbf{A}$ into triangular and diagonal matrices (ex. $\mathbf{A} = \mathbf{L}\mathbf{U}$) and solution of simpler problems.

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad \Leftrightarrow \quad \mathbf{L}\mathbf{U}\mathbf{x} = \mathbf{b} \quad \Leftrightarrow \quad \left| \begin{array}{l} \mathbf{L}\mathbf{y} = \mathbf{b} \\ \mathbf{U}\mathbf{x} = \mathbf{y} \end{array} \right.$$

  Advantages: exact solution known after a given number of operations
  Difficulties: heavy computational cost *(operations/memory)*, hard to parallelize

- **Iterative methods:** Iterative procedure to minimizing an error $\|\mathbf{x}^{(\ell)} - \mathbf{x}_{\text{ref}}\|$ and/or a residual $\|\mathbf{A}\mathbf{x}^{(\ell)} - \mathbf{b}\|$.

$$\left| \begin{array}{l} \mathbf{x}^{(0)} = \mathrm{Iter}_{(0)}\left(\mathbf{A}, \mathbf{b}\right) \\ \mathbf{x}^{(\ell+1)} = \mathrm{Iter}^{(\ell+1)}\left(\mathbf{x}^{(\ell)}, \mathbf{x}^{(\ell-1)}, \ldots, \mathbf{A}, \mathbf{b}\right), \quad \text{pour } \ell \geq 0 \end{array} \right.$$

  Advantages: limited cost per iteration *(operations/memory)*, easy to parallelize
  Difficulties: approximate solution, control of the convergence of the process

Definitions

- Lower triangular matrix: $\mathbf{L} \in \mathbb{R}^{n \times n}$ with $L_{ij} = 0$ if $i < j$
- Upper triangular matrix: $\mathbf{U} \in \mathbb{R}^{n \times n}$ with $U_{ij} = 0$ if $i > j$

$$\mathbf{L} = \begin{bmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{bmatrix} \qquad \mathbf{U} = \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{bmatrix}$$

Properties

- The determinant of a triangular matrix is the product of the diagonal elements:

$$\det(\mathbf{L}) = \prod_{i=1}^{n} L_{ii} \qquad \det(\mathbf{U}) = \prod_{i=1}^{n} U_{ii}$$

- A triangular matrix is invertible if, and only if, its diagonal elements are nonzero.

Triangular systems — Approaches by points [2/2]

Procedure to solve $\mathbf{Lx} = \mathbf{b}$ *(Forward substitution)*

Data: $\mathbf{L}$ and $\mathbf{b}$

Initialization: $\mathbf{x} \leftarrow 0$

for $i = 1, \ldots, n$ **do**

$$x_i \leftarrow \left[b_i - \sum_{j=1}^{i-1} L_{ij} x_j \right] / L_{ii}$$

end

Procedure to solve $\mathbf{Ux} = \mathbf{b}$ *(Backward substitution)*

Data: $\mathbf{U}$ et $\mathbf{b}$

Initialization: $\mathbf{x} \leftarrow 0$

for $i = n, \ldots, 1$ **do**

$$x_i \leftarrow \left[b_i - \sum_{j=i+1}^{n} U_{ij} x_j \right] / U_{ii}$$

end

Algorithmic aspects

- Cost: n^2 operations
- Weak parallelization *(computation of sums in parallel)*

Triangular systems — Approaches by blocks [1/2]

Definitions

- Lower triangular matrix by blocks: $\mathbf{L} \in \mathbb{R}^{n \times n}$ with $\mathbf{L}_{IJ} = 0$ si $I < J$
- Upper triangular matrix by blocks: $\mathbf{U} \in \mathbb{R}^{n \times n}$ with $\mathbf{U}_{IJ} = 0$ si $I > J$

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_{11} & 0 & 0 \\ \mathbf{L}_{21} & \mathbf{L}_{22} & 0 \\ \mathbf{L}_{31} & \mathbf{L}_{32} & \mathbf{L}_{33} \end{bmatrix} \qquad \mathbf{U} = \begin{bmatrix} \mathbf{U}_{11} & \mathbf{U}_{12} & \mathbf{U}_{13} \\ 0 & \mathbf{U}_{22} & \mathbf{U}_{23} \\ 0 & 0 & \mathbf{U}_{33} \end{bmatrix}$$

The diagonal blocks are square. The other blocks may not be square.

Properties

- The determinant of a triangular matrix by blocks is the product of the determinants of the diagonal blocks:

$$\det(\mathbf{L}) = \prod_{I=1}^{N} \det(\mathbf{L}_{II}) \qquad \det(\mathbf{U}) = \prod_{I=1}^{N} \det(\mathbf{U}_{II})$$

- A diagonal or triangular matrix by blocks is invertible if, and only if, the diagonal blocks are invertible.

Triangular systems — Approaches by blocks [2/2]

Procedure to solve $\mathbf{L}\mathbf{x} = \mathbf{b}$ *(Forward substitution)*

Data: $\mathbf{L}$ and $\mathbf{b}$

Initialization: $\mathbf{x} \leftarrow 0$

for $I = 1, \ldots, N$ **do**

$$\mathbf{x}_I \leftarrow \mathbf{L}_{II}^{-1} \left[\mathbf{b}_I - \sum_{J=1}^{I-1} \mathbf{L}_{IJ} \mathbf{x}_J \right]$$

end

Procedure to solve $\mathbf{U}\mathbf{x} = \mathbf{b}$ *(Backward substitution)*

Data: $\mathbf{U}$ and $\mathbf{b}$

Initialization: $\mathbf{x} \leftarrow 0$

for $I = N, \ldots, 1$ **do**

$$\mathbf{x}_I \leftarrow \mathbf{U}_{II}^{-1} \left[\mathbf{b}_I - \sum_{J=I+1}^{N} \mathbf{U}_{IJ} \mathbf{x}_J \right]$$

end

Algorithmic aspects

- Cost: N small system to solve and $N(N-1)/2$ matrix-vector products
- Better for parallel computing *(computation of matrix-vector products in parallel)*

General systems — Gaussian elimination [1/5]

Goal: Transform $\mathbf{Ax} = \mathbf{b}$ into an equivalent system $\mathbf{Ux} = \mathbf{b}'$.

Procedure

- Initialization:

$$\begin{bmatrix} A_{11}^{(1)} & A_{12}^{(1)} & \cdots & A_{1n}^{(1)} \\ A_{21}^{(1)} & A_{22}^{(1)} & \cdots & A_{2n}^{(1)} \\ \vdots & \vdots & & \vdots \\ A_{n1}^{(1)} & A_{n2}^{(1)} & \cdots & A_{nn}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ \vdots \\ b_n^{(1)} \end{bmatrix}$$

- Iteration 1:

$$\begin{bmatrix} A_{11}^{(1)} & A_{12}^{(1)} & \cdots & A_{1n}^{(1)} \\ 0 & A_{22}^{(2)} & \cdots & A_{2n}^{(2)} \\ \vdots & \vdots & & \vdots \\ 0 & A_{n2}^{(2)} & \cdots & A_{nn}^{(2)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(2)} \end{bmatrix}$$

with

$$\begin{aligned} \alpha_{i1} &= A_{i1}^{(1)} / A_{11}^{(1)} && i = 2, \ldots, n && \text{(multipliers)} \\ A_{ij}^{(2)} &= A_{ij}^{(1)} - \alpha_{i1} A_{1j}^{(1)} && i, j = 2, \ldots, n \\ b_i^{(2)} &= b_i^{(1)} - \alpha_{i1} b_1^{(1)} && i = 2, \ldots, n \end{aligned}$$

Goal: Transform $\mathbf{Ax} = \mathbf{b}$ into an equivalent system $\mathbf{Ux} = \mathbf{b}'$.

Procedure

- Iteration k:

$$\begin{bmatrix} A_{11}^{(1)} & A_{12}^{(1)} & \cdots & \cdots & \cdots & A_{1n}^{(1)} \\ 0 & A_{22}^{(2)} & & & & A_{2n}^{(2)} \\ \vdots & & \ddots & & & \vdots \\ 0 & \cdots & 0 & A_{k+1,k+1}^{(k)} & \cdots & A_{k+1,n}^{(k)} \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & A_{n,k+1}^{(k)} & \cdots & A_{n,n}^{(k)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{k+1} \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_{k+1}^{(k)} \\ \vdots \\ b_n^{(k)} \end{bmatrix}$$

- Iteration $n-1$:

$$\begin{bmatrix} A_{11}^{(1)} & A_{12}^{(1)} & \cdots & \cdots & A_{1n}^{(1)} \\ 0 & A_{22}^{(2)} & & & A_{2n}^{(2)} \\ \vdots & & \ddots & & \vdots \\ & & & \ddots & \vdots \\ 0 & & & & A_{nn}^{(n)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ \vdots \\ b_n^{(n)} \end{bmatrix}$$

General systems — Gaussian elimination [3/5]

Gaussian elimination

Data: $\mathbf{A}$ and $\mathbf{b}$

Initialization: $\mathbf{A}^{(1)} = \mathbf{A}$ and $\mathbf{b}^{(1)} = \mathbf{b}$

for $k = 1, \ldots, n-1$ **do**

  for $i = k+1, \ldots, n$ **do**

$$\alpha_{ik} = A_{ik}^{(k)} / A_{kk}^{(k)} \qquad \text{Si } A_{kk}^{(k)} \neq 0!$$

$$A_{ij}^{(k+1)} = A_{ij}^{(k)} - \alpha_{ik} A_{kj}^{(k)} \quad (j = k+1, \ldots, n)$$

$$b_i^{(k+1)} = b_i^{(k)} - \alpha_{ik} b_k^{(k)}$$

  end

end

Gaussian elimination *(Rewriting)*

Data: $\mathbf{A}$ and $\mathbf{b}$

for $k = 1, \ldots, n-1$ **do**

  for $i = k+1, \ldots, n$ **do**

$$\alpha \leftarrow A_{ik} / A_{kk} \qquad \text{Si } A_{kk} \neq 0!$$

$$A_{ij} \leftarrow A_{ij} - \alpha A_{kj} \quad (j = k+1, \ldots, n)$$

$$A_{ik} \leftarrow 0$$

$$b_i \leftarrow b_i - \alpha b_k$$

  end

end

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad \Leftrightarrow \quad \begin{bmatrix} A'_{11} & A'_{12} & A'_{13} \\ 0 & A'_{22} & A'_{23} \\ 0 & 0 & A'_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b'_1 \\ b'_2 \\ b'_3 \end{bmatrix}$$

Cost

- Gaussian elimination: $2(n-1)n(n+1)/3 + n(n-1)$ flop
- Backward substitution: n^2 flop
- Largest term: $2/3\, n^3$ flop

Conditions of use

- The method must be modified if, at one step, $A_{kk}^{(k)} = 0$.
- The method can be used without modification ...
  - if $\mathbf{A}$ is diagonally dominant per line or per column
  - if $\mathbf{A}$ is symmetric positive definite
- For the other cases $\Rightarrow$ Permutation of lines and/or columns

General systems — Gaussian elimination [5/5]

Gaussian elimination with pivoting

Data: $\mathbf{A}$ and $\mathbf{b}$

for $k = 1, \ldots, n-1$ **do**

  Find $r \in [k, \ldots, n]$ such that $|A_{rk}|$ is max. (if $\max_r |A_{rk}| = 0$, then $\mathbf{A}$ not invert.)

  Swap the r^{th} and k^{th} lines of $\mathbf{A}$ and $\mathbf{b}$

  for $i = k+1, \ldots, n$ **do**

    $\alpha \leftarrow A_{ik}/A_{kk}$

    $A_{ij} \leftarrow A_{ij} - \alpha A_{kj} \quad (j = k+1, \ldots, n)$

    $A_{ik} \leftarrow 0$

    $b_i \leftarrow b_i - \alpha b_k$

  end

end

Possible pivoting strategies

- Find $r \in [k, \ldots, n]$ such that $|A_{kr}|$ is maximum *(partial pivoting)*
- Find $r \in [k, \ldots, n]$ and $s \in [k, \ldots, n]$ such that $|A_{rs}|$ is maximum *(total pivoting)*

One can always find $r \in [k, \ldots, n]$ such that $|A_{rk}| > 0$, otherwise $\mathbf{A}$ is not invertible.

$\Rightarrow$ The Gaussian elimination with pivoting is applicable to every invertible matrix!

Generalities on matrix factorization [1/2]

Principle

- Factorize $\mathbf{A}$ into triangular matrices:

$$\underbrace{\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix}}_{\mathbf{A}} = \underbrace{\begin{bmatrix} \times & 0 & 0 \\ \times & \times & 0 \\ \times & \times & \times \end{bmatrix}}_{\mathbf{L}} \underbrace{\begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{bmatrix}}_{\mathbf{U}}$$

- Rewrite the problem with triangular matrices:

$$\mathbf{Ax} = \mathbf{b} \quad \Leftrightarrow \quad \mathbf{LUx} = \mathbf{b} \quad \Leftrightarrow \quad \begin{cases} \mathbf{Ly} = \mathbf{b} \\ \mathbf{Ux} = \mathbf{y} \end{cases}$$

  and solve these problems with forward/back substitutions.

Comments

- A priori, same cost than Gaussian elimination.
- If the system must be solved with several right-hand sides, only one factorization is necessary.

Generalities on matrix factorization [2/2]

Several factorizations

If $\mathbf{A}$ is invertible and factorizable *(to define later)*, one has:

$\mathbf{A} = \tilde{\mathbf{L}}\mathbf{U}$	(Gauss)	
$\mathbf{A} = \mathbf{L}\tilde{\mathbf{U}}$	(Gauss)	
$\mathbf{A} = \tilde{\mathbf{L}}\mathbf{D}\tilde{\mathbf{U}}$	(Gauss-Jordan)	
$\mathbf{A} = \tilde{\mathbf{L}}\mathbf{D}\tilde{\mathbf{L}}^T$	(Crout)	*Si* $\mathbf{A}$ *symétrique.*
$\mathbf{A} = \mathbf{L}\mathbf{L}^T$	(Cholesky)	*Si* $\mathbf{A}$ *symétrique définie positive (SDP).*

with

- $\mathbf{D}$ – diagonal matrix
- $\mathbf{L}$ – lower triangular matrix
- $\tilde{\mathbf{L}}$ – lower triangular matrix with unit diagonal
- $\mathbf{U}$ – upper triangular matrix
- $\tilde{\mathbf{U}}$ – upper triangular matrix with unit diagonal

Gaussian factorization ($\mathbf{A} = \tilde{\mathbf{L}}\mathbf{U}$) [1/4]

Procedure

- Initialization:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} A_{11}^{(1)} & A_{12}^{(1)} & \cdots & A_{1n}^{(1)} \\ A_{21}^{(1)} & A_{22}^{(1)} & \cdots & A_{2n}^{(1)} \\ \vdots & \vdots & & \vdots \\ A_{n1}^{(1)} & A_{n2}^{(1)} & \cdots & A_{nn}^{(1)} \end{bmatrix}$$

- Iteration 1:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ \alpha_{21} & 1 & & & 0 \\ \alpha_{31} & 0 & 1 & & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ \alpha_{n1} & 0 & \cdots & \cdots & 1 \end{bmatrix} \begin{bmatrix} A_{11}^{(1)} & A_{12}^{(1)} & \cdots & A_{1n}^{(1)} \\ 0 & A_{22}^{(2)} & \cdots & A_{2n}^{(2)} \\ 0 & A_{32}^{(2)} & \cdots & A_{3n}^{(2)} \\ \vdots & \vdots & & \vdots \\ 0 & A_{n2}^{(2)} & \cdots & A_{nn}^{(2)} \end{bmatrix}$$

with

$$\alpha_{i1} = A_{i1}^{(1)} / A_{11}^{(1)} \qquad i = 2, \ldots, n$$

$$A_{ij}^{(2)} = A_{ij}^{(1)} - \alpha_{i1} A_{1j}^{(1)} \qquad i, j = 2, \ldots, n$$

Gaussian factorization ($\mathbf{A} = \tilde{\mathbf{L}}\mathbf{U}$) [2/4]

Procedure *(continuation)*

- Iteration k:

$$\begin{bmatrix} 1 & 0 & \cdots & \cdots & \cdots & 0 \\ \alpha_{21} & \ddots & & & & 0 \\ \vdots & \ddots & 1 & & & \vdots \\ \vdots & & \vdots & 1 & & \vdots \\ \vdots & & \vdots & & \ddots & \vdots \\ \alpha_{n1} & \cdots & \alpha_{nk} & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} A_{11}^{(1)} & A_{12}^{(1)} & \cdots & \cdots & \cdots & A_{1n}^{(1)} \\ 0 & A_{22}^{(2)} & & & & A_{2n}^{(2)} \\ \vdots & & \ddots & & & \vdots \\ 0 & \cdots & 0 & A_{k+1,k+1}^{(k)} & \cdots & A_{k+1,n}^{(k)} \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & A_{n,k+1}^{(k)} & \cdots & A_{nn}^{(k)} \end{bmatrix}$$

- Iteration $n-1$:

$$\begin{bmatrix} 1 & 0 & \cdots & \cdots & 0 \\ \alpha_{21} & 1 & & & 0 \\ \vdots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & 1 & \vdots \\ \alpha_{n1} & \cdots & \cdots & \alpha_{n(n-1)} & 1 \end{bmatrix} \begin{bmatrix} A_{11}^{(1)} & A_{12}^{(1)} & \cdots & \cdots & A_{1n}^{(1)} \\ 0 & A_{22}^{(2)} & & & A_{2n}^{(2)} \\ \vdots & & \ddots & & \vdots \\ 0 & & & \ddots & \vdots \\ 0 & & & & A_{nn}^{(n)} \end{bmatrix}$$

Gaussian factorization ($\mathbf{A} = \tilde{\mathbf{L}}\mathbf{U}$) [3/4]

Gaussian factorization

Data: $\mathbf{A}$

Initialization: $\tilde{\mathbf{L}} = \mathbf{I}$ and $\mathbf{U} = \mathbf{A}$

for $k = 1, \ldots, n-1$ **do**

  for $i = k+1, \ldots, n$ **do**

    $\tilde{L}_{ik} \leftarrow U_{ik}/U_{kk}$

    $U_{ij} \leftarrow U_{ij} - \tilde{L}_{ik}U_{kj} \quad (j = k+1 \ldots n)$

    $U_{ik} \leftarrow 0$

  end

end

Si $U_{kk} \neq 0$!

Gaussian factorization *(Rewriting)*

Data: $\mathbf{A}$

for $k = 1, \ldots, n-1$ **do**

  for $i = k+1, \ldots, n$ **do**

    $A_{ik} \leftarrow A_{ik}/A_{kk}$

    $A_{ij} \leftarrow A_{ij} - A_{ik}A_{kj} \quad (j = k+1 \ldots n)$

  end

end

Si $A_{kk} \neq 0$!

Gaussian factorization ($\mathbf{A} = \tilde{\mathbf{L}}\mathbf{U}$) [4/4]

Theorem – Unicity of the Gaussian factorization ($\mathbf{A} = \tilde{\mathbf{L}}\mathbf{U}$)

The Gaussian factorization, if it exists, is unique.

Theorem – Existence of the Gaussian factorization ($\mathbf{A} = \tilde{\mathbf{L}}\mathbf{U}$) *(not necessary)*

Every matrix that is *symmetric positive definite* (SPD) is factorizable.

Theorem – Existence of the Gaussian factorization after permutation ($\mathbf{PA} = \tilde{\mathbf{L}}\mathbf{U}$)

For every invertible matrix, there exists a sequence of elementary permutations such that the permuted matrix admits a Gaussian factorization.

Permutation matrix

- An elementary permutation matrix $\mathbf{P}^{(i_1,i_2)}$ is a matrix which the application on a matrix $\mathbf{A}$ permutes the lines i_1 and i_2:

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} = \begin{bmatrix} A_{\mathbf{31}} & A_{\mathbf{32}} & A_{\mathbf{33}} & A_{\mathbf{34}} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{\mathbf{11}} & A_{\mathbf{12}} & A_{\mathbf{13}} & A_{\mathbf{14}} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix}$$

- The permutation matrix $\mathbf{P}^{(i_1,i_2)}$ is obtained by swapping the lines i_1 and i_2 and the columns i_1 and i_2 from the identity matrix.

Solution after factorization

Gaussian factorization *(Rewriting)*

Data: $\mathbf{A}$

for $k = 1, \ldots, n-1$ **do**

  for $i = k+1, \ldots, n$ **do**

    $A_{ik} \leftarrow A_{ik}/A_{kk}$ Si $A_{kk} \neq 0$!

    $A_{ij} \leftarrow A_{ij} - A_{ik}A_{kj} \quad (j = k+1 \ldots n)$

  end

end

Cost of the entire solution procedure

- Gaussian or Gauss-Jordan factorization: $2/3\, n^3$ operations
- Crout or Cholesky factorization: $1/3\, n^3$ operations
- Forward and back substitution: $2\, n^2$ operations
- Additional operations with pivoting strategies

Parallelization

- Parallelization not easy for problems with dense matrices
- Strategies are possible with blocks approaches.
- Strategies are possible with sparse matrices.

Direct methods for linear systems

Solution of triangular systems

Gaussian elimination

Matrix factorization

Performance analysis of parallel programs

Performance analysis of parallel programs

For a given sequential program,
what speedup can be expected
when it is parallelized?

Qualities for an efficient parallel program

A good parallel program is a program that ...

- minimizes the runtime,
- *(generally)* takes advantage of the compute power of the parallel machine,
- *(generally)* minimizes the communications and the waiting time.

Several tools to analyze the parallel performance

- Runtime, CPU time, computation time, communication time ...
- Strong scaling and weak scaling,
- Speedup S and efficiency E.

Performance analysis — Time

For a given parallel program, the **runtime** T is the time that elapses from the moment that a parallel computation starts to the moment that the last processor finishes execution.

The runtime depends on the slowest processor:

$$T \approx \max_p \left[T_{\text{comput}}|_p + T_{\text{comm}}|_p + T_{\text{wait}}|_p \right]$$

with, for each processor $p = 1 \dots P$,

- $T_{\text{comput}}|_p$ = time dedicated to computations
- $T_{\text{comm}}|_p$ = time dedicated to communications
- $T_{\text{wait}}|_p$ = waiting time

Reminder:

```
1 MPI_Barrier(MPI_COMM_WORLD);
2 double time1 = MPI_Wtime();
3
4 // Lots of operations for all the processus
5
6 MPI_Barrier(MPI_COMM_WORLD);
7 double time2 = MPI_Wtime();
8
9 if(myRank == 0) cout << "Duration: " << time2-time1 << endl;
```

Performance analysis — Scalability analysis [1/2]

The **scaling** or **scalability** of a parallel program is the ability to preserve the same efficiency when a larger number of processors P is used.

For a **strong scaling** analysis, P increases for a problem with a given size.
With $P\times$ more processors, can I solve a given problem $P\times$ more rapidly?

For a **weak scaling** analysis, P increases linearly with the size of the a problem.
With $P\times$ more processors, can I solve a $P\times$ larger problem with the same runtime?

In French:

- *Scaling/Scalability analysis = Analyse de scalabilité ou de passage à l'échelle*
- *Strong scaling = Scalabilité forte*
- *Weak scaling = Scalabilité faible*

Performance analysis — Speedup and efficiency

For a given parallel program, the **speedup** S is a number that measures the decrease of the runtime when P processors are used instead of 1 processor.

$$S = \frac{T_{\text{sequential}}}{T_{\text{parallel}}} \quad \in [0, P]$$

For a given parallel program, the **efficiency** E is the ratio between the actual speedup (S_{actual}) and the ideal speedup (S_{ideal}).

$$E = \frac{S_{\text{actual}}}{S_{\text{ideal}}} \quad \in [0, 1]$$

What speedup can be expected?

Illustration for a problem with different sizes:

	P	1	2	4	8	16
Normal size	S	1.0	1.9	3.1	4.8	6.2
	E	1.0	0.95	0.78	0.60	0.32
Size $\times$ 2	S	1.0	1.9	3.6	6.5	10.8
	E	1.0	0.95	0.90	0.81	0.68
Size $\times$ 4	S	1.0	1.9	3.8	7.5	14.2
	E	1.0	0.95	0.95	0.94	0.89

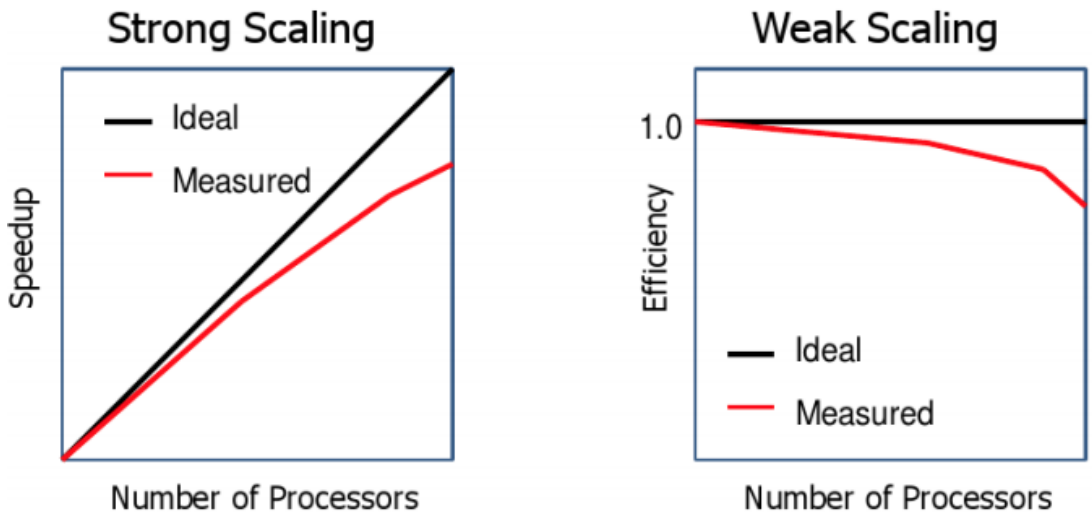
Performance analysis — Scalability analysis [2/2]

The **scaling** or **scalability** of a parallel program is the ability to preserve the same efficiency when a larger number of processors P is used.

For a **strong scaling** analysis, P increases for a problem with a given size.
With $P\times$ more processors, can I solve a given problem $P\times$ more rapidly?

For a **weak scaling** analysis, P increases linearly with the size of the a problem.
With $P\times$ more processors, can I solve a $P\times$ larger problem with the same runtime?

Presentation of weak/strong scaling analyses



Warning: the size of the problem is constant in the first case,
and it increases linearly with the number of processors in the second case.

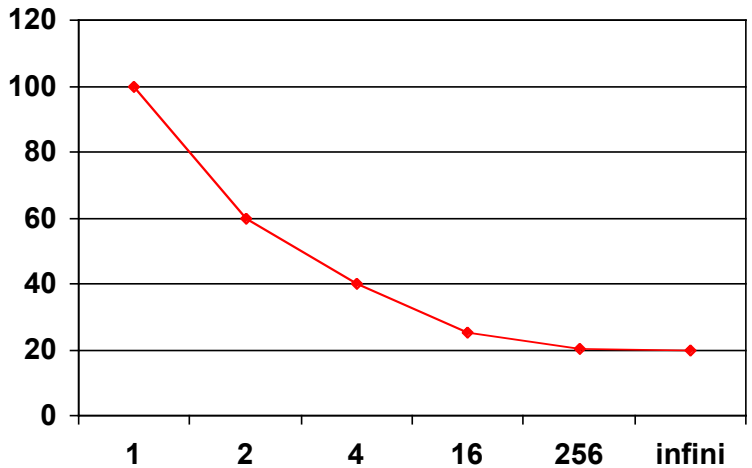
Performance analysis — Amdahl's law

Operations that must be performed sequentially prevent reaching the maximal speedup.
A more reasonable goal is given by Amdahl's law.

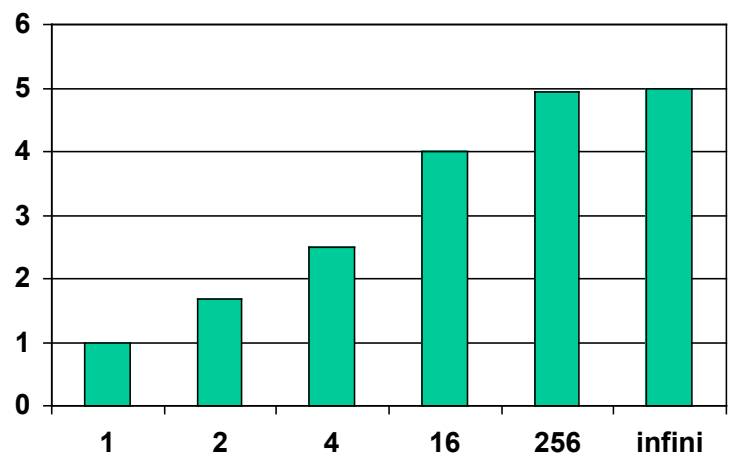
Amdahl's law

If β is the portion of the runtime of the sequential program
corresponding to operations that cannot be parallelized,
then the maximum speedup that can be reached is $S_{\text{max}} = 1/\beta$.

Illustration for a problem with $\beta = 1/5$ *and* $T_{\text{sequential}} = 100\,\text{sec}$*:*

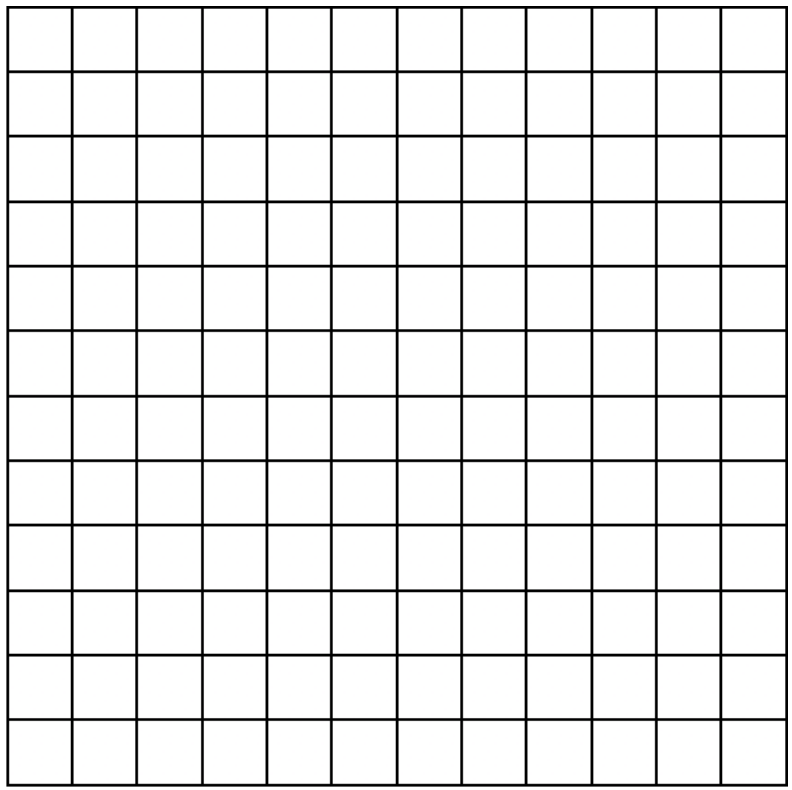


Parallel runtime T_{parallel} [sec]



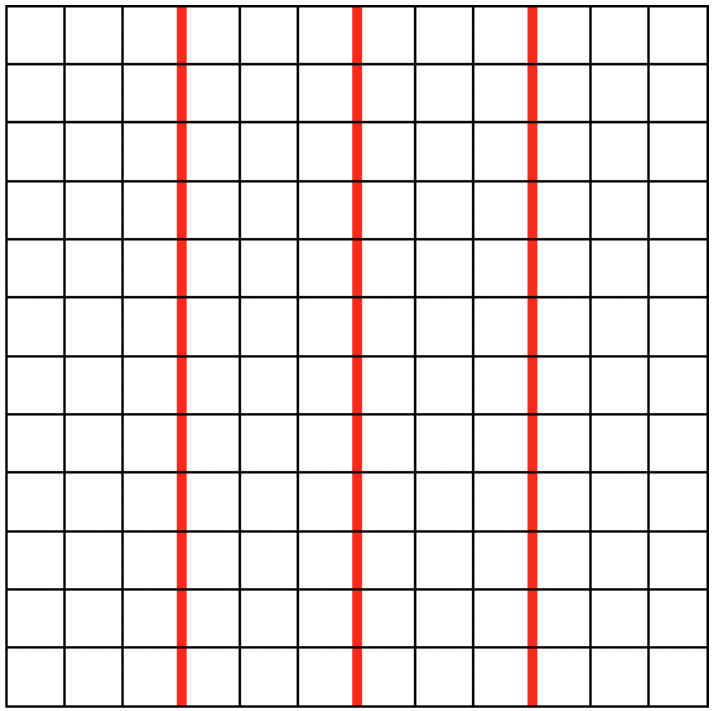
Speedup S

Here is a structured grid ...

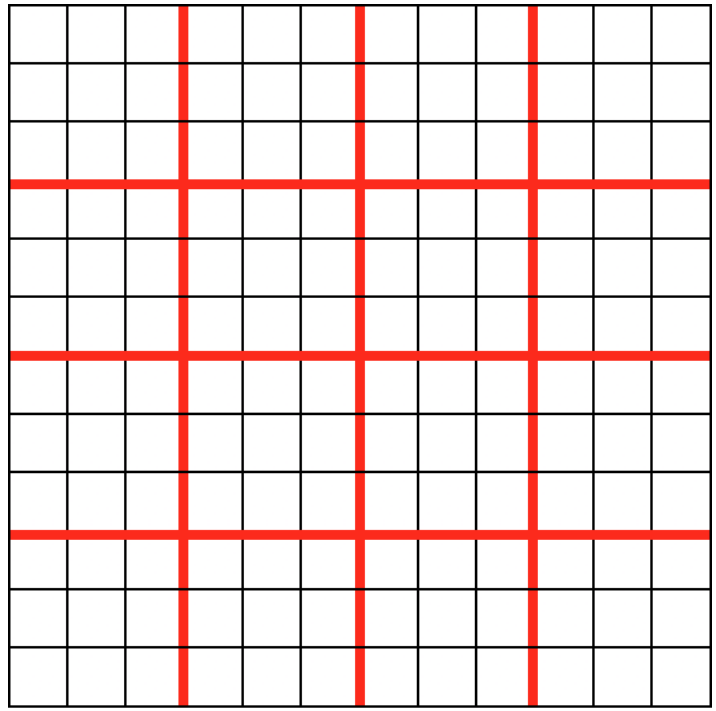


Partition for this grid?

Several partitions are possibles ...



1D partition



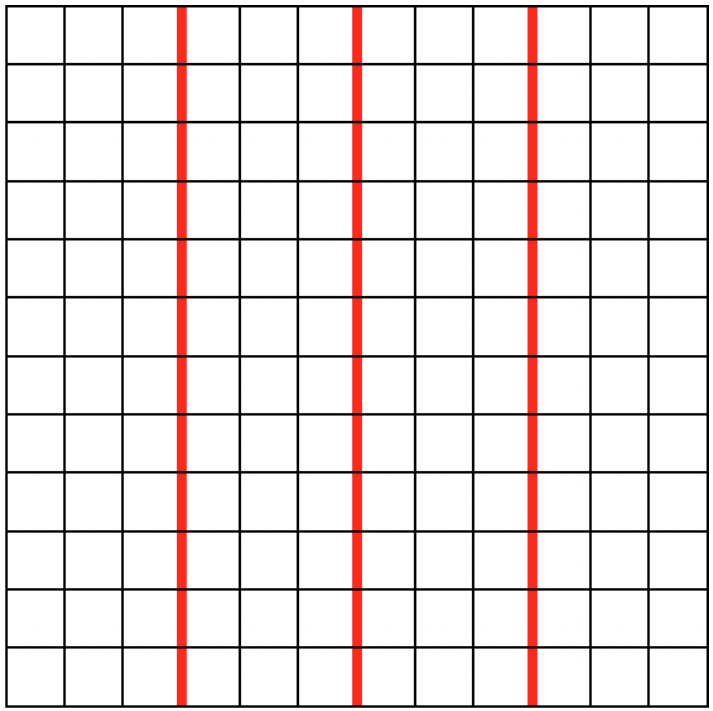
2D partition

The communication pattern and the size of the messages are different.

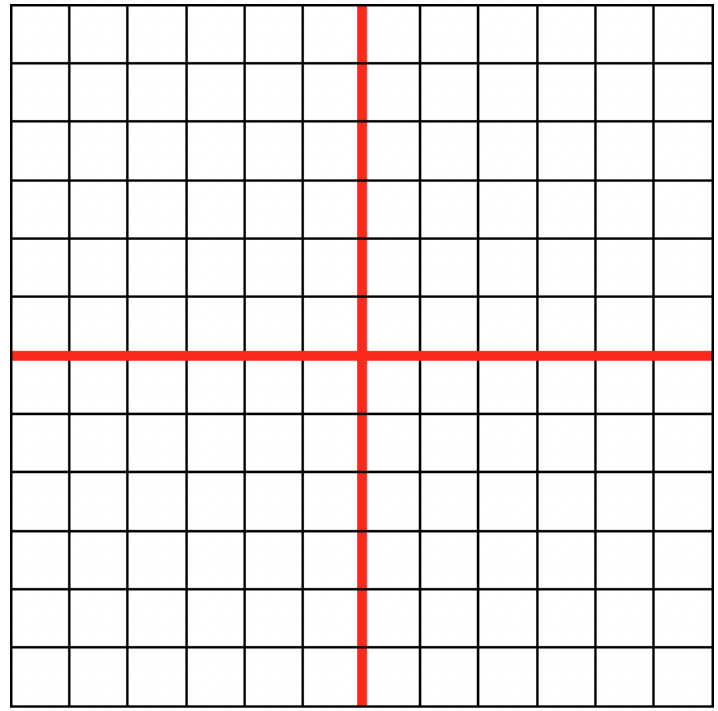
Intuitively, the grid has been partitioned by using the **method of coordinates**.

Performance analysis — Communications with a structured grid [3/4]

We consider P-partitions of a 2D grid of size $N \times N$.



1D partition with $P = 4$

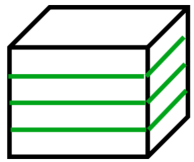
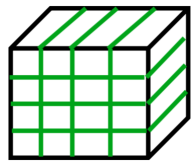
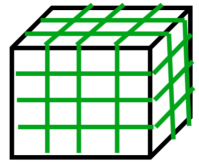


2D partition with $P = 4$

By subdomain ...	1D	2D	
Amount of transferred data	$\mathcal{O}(2N)$	$\mathcal{O}(4N/\sqrt{P})$	$\mathcal{O}(N)$
Number of operations	$\mathcal{O}(N^2/P)$	$\mathcal{O}(N^2/P)$	$\mathcal{O}(N^2)$
Ratio data / operations	$\mathcal{O}(2P/N)$	$\mathcal{O}(4\sqrt{P}/N)$	

Performance analysis — Communications with a structured grid [4/4]

We consider P-partitions of a 3D grid of size $N \times N \times N$.

By subdomain ...	 1D	 2D	 3D	
Amount of transferred data	$\mathcal{O}(2N^2)$	$\mathcal{O}(4N^2/P^{1/2})$	$\mathcal{O}(6N^2/P^{2/3})$	$\mathcal{O}(N^2)$
Number of operations	$\mathcal{O}(N^3/P)$	$\mathcal{O}(N^3/P)$	$\mathcal{O}(N^3/P)$	$\mathcal{O}(N^3)$
Ratio data / operations	$\mathcal{O}(2P/N)$	$\mathcal{O}(4P^{1/2}/N)$	$\mathcal{O}(6P^{1/3}/N)$	

Discussion

- The amount of transferred data is proportional to the **surface** of the interfaces. The number of operations is proportional to the **volume** of the subdomains.
- Increasing the number of subdomains decreases the number of operations by subdomain, but it increases the importance of the transferts. *(surface effect ↗)*
- For a large number of subdomains, it is interesting to use a partition with a high dimensionality. *(surface effect ↘)*

Summary

- **Direct solvers: Triangular systems**
  - Simple computational procedures, but not suited for parallel computing!
  - Computational cost:
    - By points: n^2 scalar operations
    - By blocks: N small systems to solve and $\mathcal{O}(N^2)$ matrix-vector products
  - Parall. comp.: by-block strategy *(dense mat.)* or ad-hoc strategy *(sparse mat.)*
- **Direct solvers: General systems**
  - Approaches:
    - Gaussian elimination $\Rightarrow$ Gives the solution for a given vector $\mathbf{b}$
    - LU factorization + Two triangular systems to solve
      $\Rightarrow$ Factorization computed once and used for any vector $\mathbf{b}$
  - For any invertible matrix $\mathbf{A}$, $\exists$ permutation matrix $\mathbf{P}$ such that $\mathbf{PA} = \tilde{\mathbf{L}}\mathbf{U}$
  - Different factorizations: $\tilde{\mathbf{L}}\mathbf{U}$, $\tilde{\mathbf{L}}\mathbf{D}\tilde{\mathbf{U}}$ (Gen), $\tilde{\mathbf{L}}\mathbf{D}\tilde{\mathbf{L}}^\top$ (Sym), $\mathbf{L}\mathbf{L}^\top$ (SDP)
  - Computational cost: $\mathcal{O}(n^3)$ operations
  - Parall. comp.: by-block strategy *(dense mat.)* or ad-hoc strategy *(sparse mat.)*
- **Performance analysis**
  - Runtime, speedup, efficiency
  - Strong/weak scaling analysis
  - Amdahl's law
  - Surface/volume effect

Resources

- *Méthodes Numériques : Algorithmes, analyse et applications*
  A. Quarteroni, R. Sacco, F. Saleri (2007), Springer
- *Calcul scientifique parallèle*
  F. Magoulès et F.-X. Roux (2017), Dunod
- *Calcul scientifique parallèle*
  P. Ciarlet et E. Jamelot, polycopié de cours