Sparse Linear Systems and parallel iterative methods Lesson 8.

Adriano FESTA

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica, L'Aquila

DISIM, L'Aquila, 07.05.2019

adriano.festa@univaq.it

Class outline

- Poisson Equation
- Tridiagonal Systems
- General banded matrices
- Domain Decomposition Methods
- Heat Equation

Motivation: Poisson Equation

- As a typical example of an elliptic partial differential equation we consider the Poisson equation with Dirichlet boundary conditions.
- This equation is often called the model problem since its structure is simple but the numerical solution is very similar to many other more complicated partial differential equations,
- The two-dimensional Poisson equation has the form

$$-\Delta u(x, y) = f(x, y) \quad \text{for all } (x, y) \in \Omega$$

  with domain $\Omega \in \mathbb{R}^2$.
- The function $u : \mathbb{R}^2 \to \mathbb{R}$ is the unknown solution function and the function $f : \mathbb{R}^2 \to \mathbb{R}$ is the right-hand side, which is continuous in Ω and its boundary.
- The operator Δ is the two-dimensional Laplace operator

$$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

Motivation: Poisson Equation

- Using this notation, the Poisson equation can also be written as

$$-\frac{\partial^2}{\partial x^2}u(x,y) - \frac{\partial^2}{\partial y^2}u(x,y) = f(x,y) \quad \text{for all } (x,y) \in \Omega.$$

- A model problem uses the unit square $\Omega = (0,1) \times (0,1)$and assumes a Dirichlet boundary condition

$$u(x,y) = \phi(x,y) \text{for all } (x,y) \in \partial\Omega$$

  where ϕ is a given function and $\partial\Omega$ is the boundary of domain Ω, which is $\partial\Omega = \{(x,y)|0 \leq x \leq 1, y = 0 \text{ or } y = 1\} \cup \{(x,y)|0 \leq y \leq 1, x = 0 \text{ or } x = 1\}$. The boundary condition uniquely determines the solution u of the model problem.
- An example of the Poisson equation from electrostatics is the equation

$$\Delta u = -\frac{\rho}{\varepsilon_0}$$

  where ρ is the charge density, ε_0 is a constant, and u is the electrical potential created by the charge.

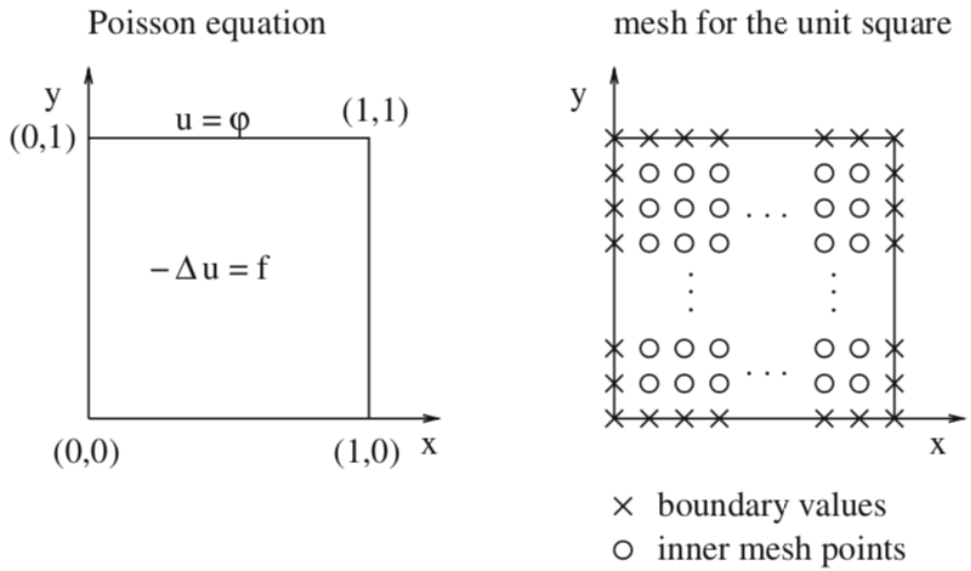
Poisson Equation: Discretization

- For the numerical solution of equation $-\Delta u(x, y) = f(x, y)$, the method of finite differences can be used, which is based on a discretization of the domain $\Omega \cup \partial\Omega$
- The discretization is given by a regular mesh with $N + 2$ mesh points in x-direction and in y-direction, where N points are in the inner part and 2 points are on the boundary. The distance between points in the x- or y-direction is $h = 1/(N + 1)$. The mesh points are

$$(x_i, y_j) = (ih, jh) \quad \text{for } i, j = 0, 1, ..., N + 1.$$

- The points on the boundary are the points with $x_0 = 0$, $y_0 = 0$, $x_{N+1} = 1$, or $y_{N+1} = 1$.

Poisson Equation: Discretization



Left: Poisson equation with Dirichlet boundary condition on the unit square $\Omega = (0, 1) \times (0, 1)$. *Right*: The numerical solution discretizes the Poisson equation on a mesh with equidistant mesh points with distance $1/(N + 1)$. The mesh has N^2 inner mesh points and additional mesh points on the boundary

- The unknown solution function u is determined at the points (x_i, y_j) of this mesh, which means that values $u_{ij} := u(x_i, y_j)$ for $i, j = 0, 1, ..., N + 1$ are to be found.

Poisson Equation: FD Discretization

- For the inner part of the mesh, these values are determined by solving a linear equation system with N^2 equations.
- For each mesh point (x_i, y_j), $i, j = 1, ..., N$, a Taylor expansion is used for the x or y-direction.
- The Taylor expansion in x-direction is

$$u(x_i + h, y_j) = u(x_i, y_j) + h \cdot u_x(x_i, y_j) + \frac{h^2}{2} u_{xx}(x_i, y_j) + \frac{h^3}{6} u_{xxx}(x_i, y_j) + O(h^4),$$

$$u(x_i - h, y_j) = u(x_i, y_j) - h \cdot u_x(x_i, y_j) + \frac{h^2}{2} u_{xx}(x_i, y_j) - \frac{h^3}{6} u_{xxx}(x_i, y_j) + O(h^4).$$

where u_x denotes the partial derivative in x -direction (i.e., $u_x = \partial u / \partial x$) and u_{xx} denotes the second partial derivative in x-direction (i.e., $u_{xx} = \partial^2 u / \partial x^2$).

Poisson Equation: FD Discretization

- Adding these two Taylor expansions results in

$$u(x_i + h, y_j) + u(x_i - h, y_j) = 2u(x_i, y_j) + h^2 u_{xx}(x_i, y_j) + O(h^4).$$

- Analogously, the Taylor expansion for the y-direction can be used to get

$$u(x_i, y_j + h) + u(x_i, y_j - h) = 2u(x_i, y_j) + h^2 u_{yy}(x_i, y_j) + O(h^4).$$

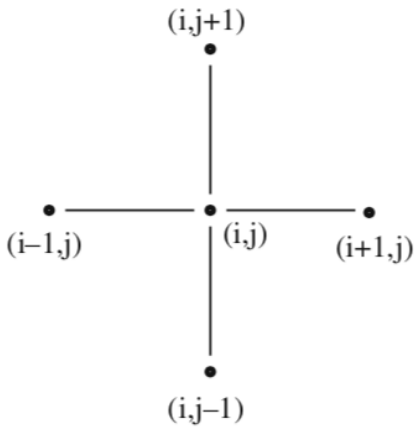
- From the last two equations, an approximation for the Laplace operator $\Delta u = u_{xx} + u_{yy}$ at the mesh points can be derived

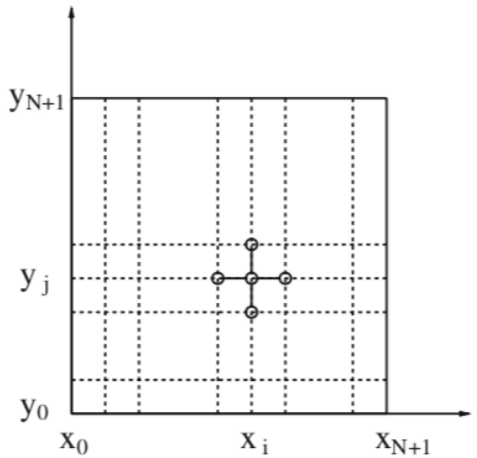
$$\Delta u(x_i, y_j) = -\frac{1}{h^2}(4u_{ij} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1}),$$

  where the higher order terms $O(h^4)$ are neglected.

Poisson Equation: Discretization

Five-point stencil resulting from the discretization of the Laplace operator with a finite difference scheme. The computation at one mesh point uses values at the four neighbor mesh points





- This pattern is known as five-point stencil. Using the approximation of Δu and the notation $f_{ij} := f(x_i, y_j)$ for the values of the right-hand side, the discretized Poisson equation or five-point formula results:

$$\frac{1}{h^2}(4u_{ij} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1}) = f_{ij} \quad \text{for} \quad 1 \leq i, j \leq N.$$

- For the points on the boundary, the values of u_{ij} result from the boundary condition and are given by

$$u_{ij} = \phi(x_i, y_j) \quad \text{for } (i,j) \in \partial\Omega.$$

Poisson Equation: linear system

- The five-point formula including boundary values represents a linear equation system with N^2 equations, N^2 unknown values, and a coefficient matrix $A \in R^{N^2 \times N^2}$.
- In order to write the equation system with boundary values in matrix form $Az = d$, the N^2 unknowns $u_{ij}, i, j = 1, ..., N$, are arranged in row-oriented order in a one-dimensional vector z of size $n = N^2$ which has the form

$$z = (u_{11}, u_{21}, ..., u_{N1}, u_{12}, u_{22}, ..., u_{N2}, ..., u_{1N}, u_{2N}, ..., u_{NN}).$$

The mapping of values u_{ij} to vector elements z_k is

$$z_k := u_{ij} \quad \text{with } k = i + (j-1)N \quad \text{for } i, j = 1, ..., N.$$

Poisson Equation: linear system

- Using the vector z, the five-point formula has the form

$$\frac{1}{h^2}(4z_{i+(j-1)N} - z_{i+(j-1)N-1} - z_{i+(j-1)N+1} - z_{i+jN} - z_{i+(j-2)N}) = d_{i+(j-1)N}$$

  for $1 \leq i, j \leq N$. with

$$d_{i+(j-1)N} = f_{ij}$$

  one-dimensional vector resulting in the corresponding mapping of f.
- Replacing the indices by $k = i + (j-1)N$ we obtain the easier form

$$\frac{1}{h^2}(4z_k - z_{k-1} - z_{k+1} - z_{k+N} - z_{k-N}) = d_k$$

  for $1 \leq k \leq N^2$.
- Thus, the entries in row k of the coefficient matrix contain five entries which are $a_{kk} = 4$ and $a_{k,k+1} = a_{k,k-1} = a_{k,k+N} = a_{k,k-N} = -1$.

- The building of the vector d and the coefficient matrix $A = (a_{ij})$, $i, j = 1, ..., N^2$, can be performed by the following algorithm.
- The loops over i and j , $i, j = 1, ..., N$, visit the mesh points (i, j) and build one row of the matrix A of size $N^2 \times N^2$.
- When $(i, j)$is an inner point of the mesh, i.e., $i, j \neq 1, N$, the corresponding row of A contains five elements at the position $k, k+1, k-1, k+N, k-N$ for $k = i + (j-1)N$.
- When (i, j) is at the boundary of the inner part, i.e., $i = 1$, $j = 1$, $i = N$, or $j = N$, the boundary values for ϕ are used.

```
/* Algorithm for building the matrix A and the vector d */
Initialize all entries of A with 0;
for (j = 1; j <= N; j++)
    for (i = 1; i <= N; i++) {
        /* Build d_k and row k of A with k = i + (j - 1)N */
        k = i + (j - 1) · N;
        a_{k,k} = 4/h^2;
        d_k = f_{ij};
        if (i > 1) a_{k,k-1} = -1/h^2 else d_k = d_k + 1/h^2 φ(0, y_j);
        if (i < N) a_{k,k+1} = -1/h^2 else d_k = d_k + 1/h^2 φ(1, y_j);
        if (j > 1) a_{k,k-N} = -1/h^2 else d_k = d_k + 1/h^2 φ(x_i, 0);
        if (j < N) a_{k,k+N} = -1/h^2 else d_k = d_k + 1/h^2 φ(x_i, 1);
    }
```

Poisson Equation: linear system

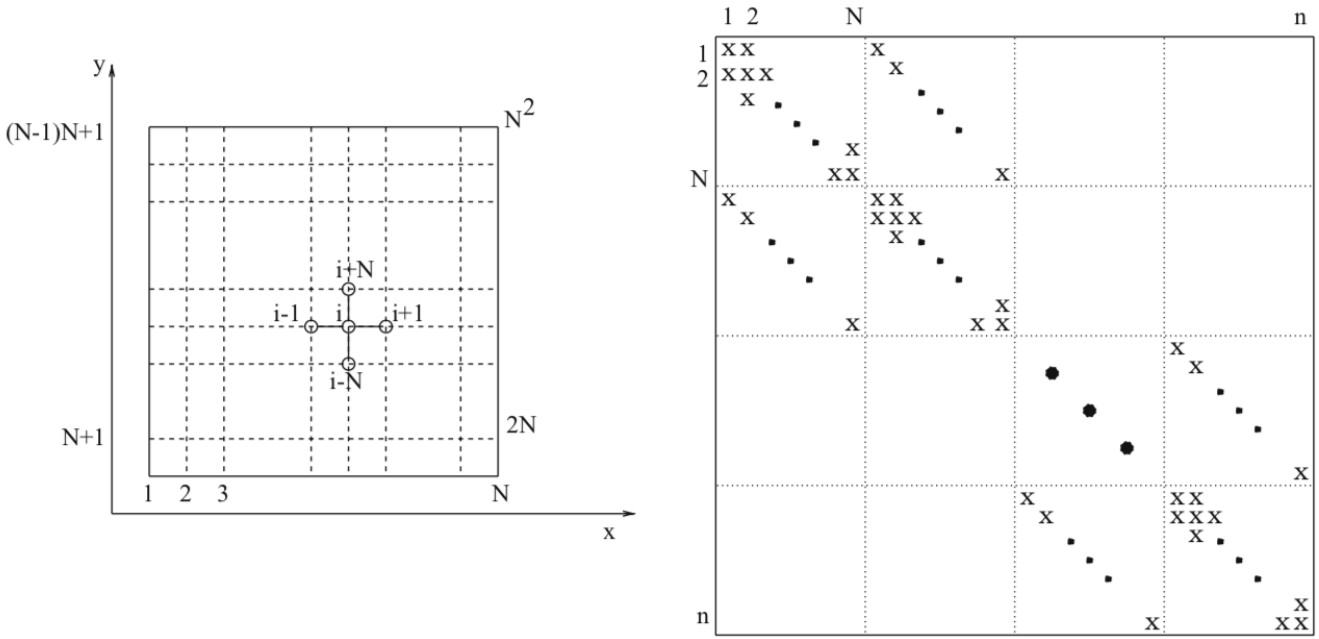
- The linear equation system resulting from this algorithm has the structure

$$\frac{1}{h^2}\begin{pmatrix} B & -I & & 0 \\ -I & B & \ddots & \\ & \ddots & \ddots & -I \\ 0 & & -I & B \end{pmatrix} \cdot z = d$$

where I denotes the $N \times N$ unit matrix, which has the value 1 in the diagonal elements and the value 0 in all other entries. The matrix B has the structure

$$B = \begin{pmatrix} 4 & -1 & & 0 \\ -1 & 4 & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 4 \end{pmatrix}$$

Poisson Equation: sparse structure



Rectangular mesh in the $x-y$ plane of size $N \times N$ and the $n \times n$ coefficient matrix with $n = N^2$ of the corresponding linear equation system of the five-point formula. The sparsity structure of the matrix corresponds to the adjacency relation of the mesh points. The mesh can be considered as adjacency graph of the non-zero elements of the matrix

- In summary, this formula represent a linear equation system with a sparse coefficient matrix, which has non-zero elements in the main diagonal and its direct neighbours as well as in the diagonals in distance N .
- Thus, the linear equation system resulting from the Poisson equation has a banded structure, which should be exploited when solving the system.

Tridiagonal Systems

- For the solution of a linear equation system $Ax = y$ with a banded or tridiagonal coefficient matrix $A \in R^{n \times n}$, specific solution methods can exploit the sparse matrix structure.
- A matrix $A = (a_{ij})_{i,j=1,\ldots,n} \in \mathbb{R}^{n \times n}$ is called banded when its structure takes the form of a band of non-zero elements around the principal diagonal.
- More precisely, this means a matrix A is a banded matrix if there exists $r \in N$, $r \leq n$, with $a_{ij} = 0$ for $|i - j| > r$.
- The number r is called the semi-bandwidth of A. For $r = 1$ a banded matrix is called tridiagonal matrix. We first consider the solution of tridiagonal systems which are linear equation systems with tridiagonal coefficient matrix.

Gaussian Elimination for Tridiagonal Systems

For the solution of a linear equation system $Ax = y$ with tridiagonal matrix A, the Gaussian elimination can be used. Step k of the forward elimination (without pivoting) results in the following computations:

1. Compute $l_{ik} := a_{ik}^{(k)} / a_{kk}^{(k)}$ for $i = k+1, ..., n$.
2. Subtract l_{ik} times the k-th row from the rows $i = k+1, ..., n$, i.e., compute

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik} \cdot a_{kj}^{(k)} \text{ for } k \leq j \leq n \text{ and } k < i \leq n.$$

The vector y is changed analogously.

Because of the tridiagonal structure of A, all matrix elements a_{ik} with $i \geq k+2$ are zero elements, i.e., $a_{ik} = 0$. Thus, in each step k of the Gaussian elimination only one elimination factor $l_{k+1} := l_{k+1,k}$ and only one row with only one new element have to be computed.

Gaussian Elimination for Tridiagonal Systems

Using the notation

$$A = \begin{pmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{pmatrix}$$

for the matrix elements and starting with $u1 = b1$, these computations are

$$\begin{aligned} l_{k+1} &= a_{k+1}/u_k, \\ u_{k+1} &= b_{k+1} - l_{k+1} \cdot c_k. \end{aligned}$$

After $n-1$ steps we obtain a LU decomposition $A = LU$ of matrix A with

$$L = \begin{pmatrix} 1 & 0 & & 0 \\ l_2 & 0 & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & l_n & 1 \end{pmatrix}, \quad U = \begin{pmatrix} u_1 & c_1 & & 0 \\ 0 & u_2 & \ddots & \\ & \ddots & u_{n-1} & c_{n-1} \\ 0 & & 0 & u_n \end{pmatrix}$$

Gaussian Elimination for Tridiagonal Systems

The right-hand side y is transformed correspondingly according to

$$\bar{y}_{k+1} = y_{k+1} - l_{k+1} \cdot \bar{y}_k.$$

The solution x is computed from the upper triangular matrix U by a backward substitution, starting with $x_n = \bar{y}_n / u_n$ and solving the equations $u_i x_i + c_i x_{i+1} = \bar{y}_i$ one after another resulting in

$$x_i = \frac{\bar{y}_i}{u_i} - \frac{c_i}{u_i} x_{i+1} \text{ for } i = n-1, ..., 1.$$

- The computational complexity of the Gaussian elimination is reduced to O(n) for tridiagonal systems.
- However, the elimination phase computing l_k and u_k since the computation of l_{k+1} depends on u_k and the computation of u_{k+1} depends on l_{k+1}.
- Thus, in this form the Gaussian elimination or LU decomposition has to be computed sequentially and is not suitable for a parallel implementation.

Recursive Doubling for Tridiagonal Systems

- An alternative approach for solving a linear equation system with tridiagonal matrix is the method of recursive doubling or cyclic reduction.
- The methods of recursive doubling and cyclic reduction also use elimination steps but contain potential parallelism. Both techniques can be applied if the coefficient matrix is either symmetric and positive definite or diagonal dominant.
- The elimination steps in both methods are applied to linear equation systems $Ax = y$ with the matrix structure

$$\begin{array}{llllll} & b_1x_1+ & c_1x_2 & = & y_1, & \\ a_ix_{i-1}+ & b_ix_i+ & c_ix_{i+1} & = & y_i & \text{for } i = 2, \ldots, n-1, \\ a_nx_{n-1}+ & b_nx_n & & = & y_n. & \end{array}$$

- the method uses two equations $i-1$ and $i+1$ to eliminate the variables x_{i-1} and x_{i+1} from equation i.
- This results in a new equivalent equation system with a coefficient matrix with three non-zero diagonals where the diagonals are moved to the outside.

Recursive Doubling for Tridiagonal Systems

- Recursive doubling and cyclic reduction can be considered as two implementation variants for the same numerical idea.
- The implementation of recursive doubling repeats the elimination step, which finally results in a matrix structure in which only the elements in the principal diagonal are non-zero and the solution vector x can be computed easily.
- Cyclic reduction is a variant of recursive doubling which also eliminates variables using neighboring rows. But in each step the elimination is only applied to half of the equations and, thus, less computations are performed.
- On the other hand, the computation of the solution vector x requires a substitution phase.

Recursive Doubling for Tridiagonal Systems

- Recursive doubling considers three neighboring equations $i-1$, i, $i+1$ of the equation system $Ax = y$ with coefficient matrix A tridiagonal. These equations are

$$\begin{array}{lllll} a_{i-1}x_{i-2} & +b_{i-1}x_{i-1} & +c_{i-1}x_i & & = y_{i-1}, \\ & a_i x_{i-1} & +b_i x_i & +c_i x_{i+1} & = y_i, \\ & & a_{i+1}x_i & +b_{i+1}x_{i+1} \quad +c_{i+1}x_{i+2} & = y_{i+1}, \end{array}$$

- Equation $i-1$ is used to eliminate x_{i-1} from the i-th equation and equation $i+1$ is used to eliminate x_{i+1} from the i-th equation. This is done by reformulating equations $i-1$ and $i+1$ to

$$x_{i-1} = \frac{y_{i-1}}{b_{i-1}} - \frac{a_{i-1}}{b_{i-1}}x_{i-2} - \frac{c_{i-1}}{b_{i-1}}x_i,$$

$$x_{i+1} = \frac{y_{i+1}}{b_{i+1}} - \frac{a_{i+1}}{b_{i+1}}x_i - \frac{c_{i+1}}{b_{i+1}}x_{i+2},$$

and inserting those descriptions of x_{i-1} and x_{i+1} into equation i.

Recursive Doubling for Tridiagonal Systems

- The resulting new equation i is

$$a^{(1)}x_{i-2} + b^{(1)}x_i + c^{(1)}x_{i+2} = y^{(1)}$$

with coefficients

$$\begin{aligned} a_i^{(1)} &= \alpha_i^{(1)} \cdot a_{i-1}, \\ b^{(1)} &= b_i + \alpha_i^{(1)} \cdot c_{i-1} + \beta_i^{(1)} \cdot a_{i+1}, \\ c^{(1)} &= \beta_i^{(1)} \cdot c_{i+1}, \\ y_i^{(1)} &= y_i + \alpha_i^{(1)} \cdot y_{i-1} + \beta_i^{(1)} \cdot y_{i+1}, \end{aligned}$$

and

$$\begin{aligned} \alpha^{(1)} &:= -a_i/b_{i-1}, \\ \beta^{(1)} &:= -c_i/b_{i+1}. \end{aligned}$$

For the special cases $i = 1, 2, n-1, n$, the coefficients are given by

$$\begin{aligned} b^{(1)} = b_1 + \beta^{(1)} \cdot a_2, \quad y^{(1)} = y_1 + \beta^{(1)} \cdot y_2, \\ b^{(1)} = b_n + \alpha^{(1)} \cdot c_{n-1}, \quad y^{(1)} = b_n + \alpha_n^{(1)} \cdot y_{n-1}, \\ a_1^{(1)} = a_2^{(1)} = 0, \text{ and } c_{n-1}^{(1)} = c_n^{(1)} = 0. \end{aligned}$$

Recursive Doubling for Tridiagonal Systems

- We reduced then to a linear equation system $A^{(1)}x = y^{(1)}$ with a coefficient matrix

$$A^{(1)} = \begin{pmatrix} b_1^{(1)} & 0 & c_1^{(1)} & & & 0 \\ 0 & b_2^{(1)} & 0 & c_2^{(1)} & & \\ a_3^{(1)} & 0 & b_3^{(1)} & \ddots & \ddots & \\ & a_4^{(1)} & \ddots & \ddots & \ddots & c_{n-2}^{(1)} \\ & & \ddots & \ddots & \ddots & 0 \\ 0 & & & a_n^{(1)} & 0 & b_n^{(1)} \end{pmatrix}$$

Comparing the structure of $A^{(1)}$ with the structure of A, it can be seen that the diagonals are moved to the outside.

Recursive Doubling for Tridiagonal Systems

- In the next step, this method is applied to the equations $i-2$, i, $i+2$ of the equation system $A^{(1)}x = y^{(1)}$ for $i = 5, 6, ..., n-4$. Equation $i-2$ is used to eliminate x_{i-2} from the ith equation and equation $i+2$ is used to eliminate x_{i+2} from the ith equation. This results in a new ith equation

$$a^{(2)}x_{i-4} + b^{(2)}x_i + c^{(2)}x_{i+4} = y^{(2)},$$

  which contains the variables x_{i-4}, x_i, and x_{i+4}.
- The cases $i = 1, ..., 4, n-3, ..., n$ are treated separately as shown for the first elimination step.

Recursive Doubling for Tridiagonal Systems

- Altogether a next equation system $A^{(2)}x = y^{(2)}$ results in which the diagonals are further moved to the outside. The structure of $A(2)$ is

$$A^{(1)} = \begin{pmatrix} b_1^{(2)} & 0 & 0 & 0 & c_1^{(2)} & & & 0 \\ 0 & b_2^{(2)} & & & & c_2^{(2)} & & \\ 0 & & \ddots & & & & \ddots & \\ 0 & & & \ddots & & & & c_{n-4}^{(2)} \\ a_5^{(2)} & & & & \ddots & & & 0 \\ & a_6^{(2)} & & & & \ddots & & 0 \\ & & \ddots & & & & \ddots & 0 \\ 0 & & & a_n^{(2)} & 0 & 0 & 0 & b_n^{(2)} \end{pmatrix}$$

Recursive Doubling for Tridiagonal Systems

- The following steps of the recursive doubling algorithm apply the same method to the modified equation system of the last step. Step k transfers the side diagonals $2^k - 1$ positions away from the main diagonal, compared to the original coefficient matrix. This is reached by considering equations $i - 2^{k-1}$, i, $i + 2^{k-1}$:

$$\begin{array}{lllll} a^{(k-1)}_{i-2^{k-1}} x_{i-2^k} & +b^{(k-2^{k-1})}_{i-2^{k-1}} x_{i-1} & +c^{(k-1)}_{i-2^{k-1}} x_i & & = y_{i-2^{k-1}}, \\ & a^{(k-1)}_i x_{i-2^{k-1}} & +b^{(k-1)}_i x_i & +c^{(k-1)}_i x_{i+2^{k-1}} & = y_i, \\ & & a^{(k-1)}_{i+2^{k-1}} x_i & +b^{(k-1)}_{i+2^{k-1}} x_{i+2^{k-1}} & +c^{(k-1)}_{i+2^{k-1}} x_{i+2^k} = y_{i+2^{k-1}}, \end{array}$$

- Equation $i - 2^{k-1}$ is used to eliminate $x_{i-2^{k-1}}$ from the ith equation and equation $i + 2^{k-1}$ is used to eliminate $x_{i+2^{k-1}}$ from the ith equation.
- Again, the elimination is performed by computing the coefficients for the next equation system.

Recursive Doubling for Tridiagonal Systems

- After $N = \lceil \log n \rceil$ steps, the original matrix A is transformed into a diagonal matrix $A^{(N)}$

$$A^{(N)} = diag(b_1^{(N)}, ..., b_n^{(N)})$$

in which only the main diagonal contains non-zero elements. The solution x of the linear equation system can be directly computed using this matrix and the correspondingly modified vector $y^{(N)}$:

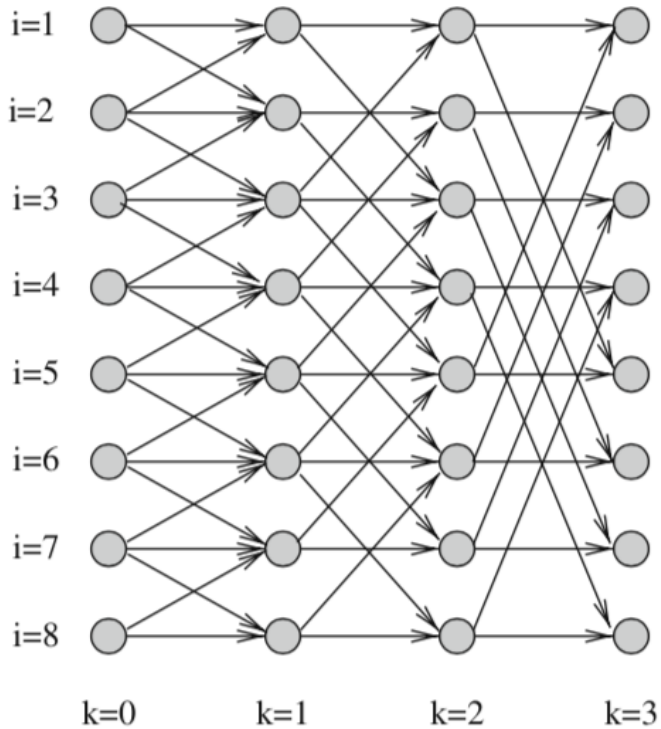
$$x_i = y_i^{(N)} / b_i^{(N)} \text{ for } i = 1, 2, ..., n.$$

Recursive Doubling for Tridiagonal Systems

- To summarize, the recursive doubling algorithm consists of two main phases:
  1. Elimination phase: Compute the values $a_i^{(k)}$, $b_i^{(k)}$, $c_i^{(k)}$, and $y_i^{(k)}$ for $k = 1, \ldots, \lceil \log n \rceil$ and $i = 1, \ldots, n$.
  2. Solution phase: Compute $xi = y_i^{(N)} / b_i^{(N)}$ for $i = 1, \ldots, n$ with $N = \log n$.

  The first phase consists of $\lceil \log \rceil$ steps where in each step $O(n)$ values are computed.
- The sequential asymptotic runtime of the algorithm is therefore $O(n \cdot \log n)$ which is asymptotically slower than the $O(n)$ runtime for the Gaussian elimination approach described earlier.
- The advantage is that the computations in each step of the elimination and the substitution phase are independent and can be performed in parallel.

Recursive Doubling for Tridiagonal Systems



Dependence graph for the computation steps of the recursive doubling algorithm in the case of three computation steps and eight equations. The computations of step k are shown in column k of the illustration. Column k contains one node for each equation i, thus representing the computation of all coefficients needed in step k. Column 0 represents the data of the coefficient matrix of the linear system. An edge from a node i in step k to a node j in step $k+1$ means that the computation at node j needs at least one coefficient computed at node i

Cyclic Reduction for Tridiagonal Systems

- The recursive doubling algorithm offers a large degree of potential parallelism but has a larger computational complexity than the Gaussian elimination caused by computational redundancy. The cyclic reduction algorithm is a modification of recursive doubling which reduces the amount of computations to be performed.
- In each step, half the variables in the equation system are eliminated which means that only half of the values $a_i^{(k)}$, $b_i^{(k)}$, $c_i^{(k)}$, and $y_i^{(k)}$ are computed.
- A substitution phase is needed to compute the solution vector x.

Cyclic Reduction for Tridiagonal Systems

- The elimination and the substitution phases of cyclic reduction are described by the following two phases:
  1. Elimination phase: For $k = 1, \ldots, \lfloor \log n \rfloor$ compute $a_i^{(k)}$, $b_i^{(k)}$, $c_i^{(k)}$, and $y_i^{(k)}$ with $i = 2^k, \ldots, n$ and step size 2^k. The number of equations of the tridiagonal form is reduced by a factor of 1/2 in each step. In step $k = \lfloor \log n \rfloor$ there is only one equation left for $i = 2^N$ with $N = \lfloor \log n \rfloor$.
  2. Substitution phase: For $k = \lfloor \log n \rfloor \ldots, 0$ compute x_i:

$$x_i = \frac{y_i^{(k)} - a_i^{(k)} \cdot x_{i-2^k} - c_i^{(k)} \cdot x_{i+2^k}}{b_i^{(k)}}.$$

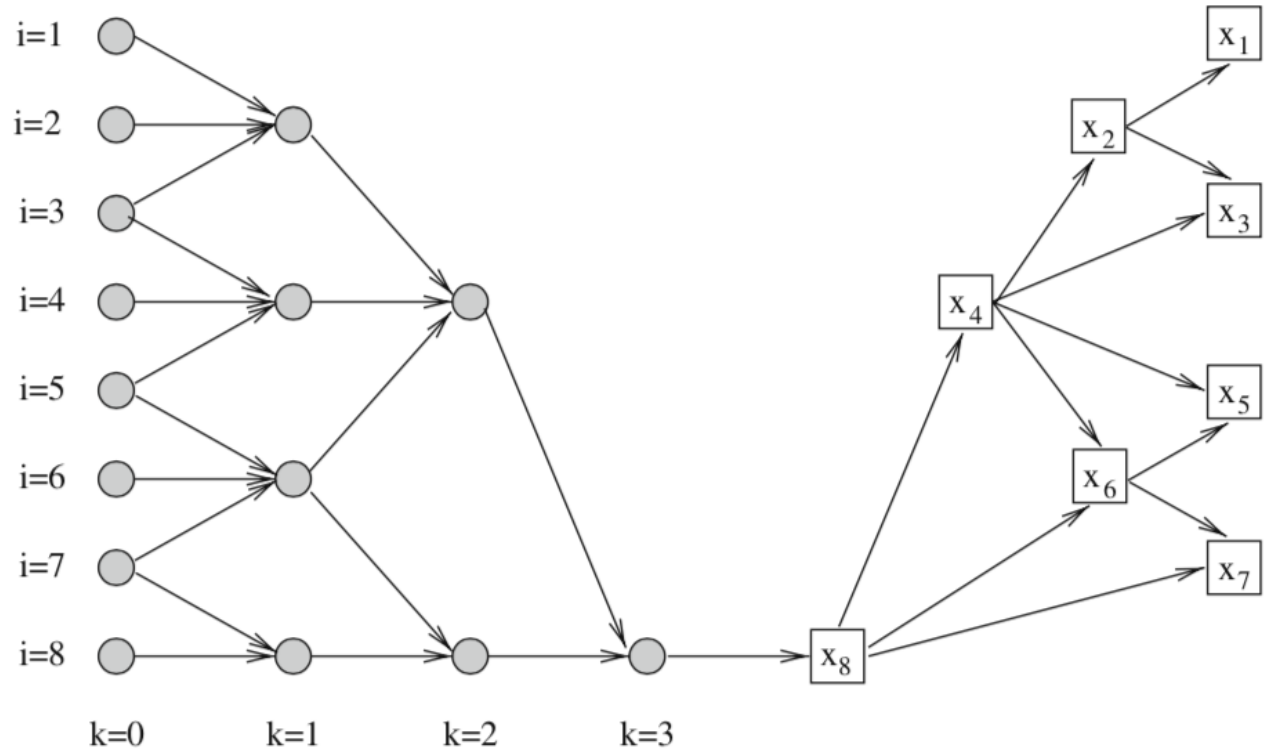
Cyclic Reduction for Tridiagonal Systems

- In each computation step k, $k = 1, ..., \lfloor \log n \rfloor$, of the elimination phase, there are $n/2^k$ nodes representing the computations for the coefficients of one equation. This results in

$$\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + ... + \frac{n}{2^k} = n \cdot \sum_{i=1}^{\lfloor \log n \rfloor} \frac{1}{2^i} \leq n$$

  computation nodes with $N = \lfloor \log n \rfloor$ and, therefore, the execution time of cyclic reduction is $O(n)$.
- Thus, the computational complexity is the same as for the Gaussian elimination; however, the cyclic reduction offers potential parallelism.

Cyclic Reduction for Tridiagonal Systems



Dependence graph illustrating the dependencies between neighboring computation steps of the cyclic reduction algorithm for the case of three computation steps and eight equations in analogy to the representation in Fig. 7.10. The first four columns represent the computations of the coefficients. The last columns in the graph represent the computation of the solution vector x in the second phase of the cyclic reduction algorithm, see (7.27)

Parallel Implementation of Cyclic Reduction

- We consider a parallel algorithm for the cyclic reduction for p processors. For the description of the phases we assume $n = p \cdot q$ for $q \in \mathcal{N}$ and $q = 2^Q$ for $Q \in \mathcal{N}$.
- Each processor stores a block of rows of size q, i.e., processor P_i stores the rows of A with the numbers $(i-1)q+1, ..., i \cdot q$ for $1 \leq i \leq p$.
- We describe the parallel algorithm with data exchange operations that are needed for an implementation with a distributed address space. As data distribution a row-blockwise distribution of the matrix A is used to reduce the interaction between processors as much as possible.
- The parallel algorithm for the cyclic reduction comprises three phases: the elimination phase stopping earlier than described above, an additional recursive doubling phase, and a substitution phase.

Cyclic Reduction for Tridiagonal Systems

- **Phase 1:** Parallel reduction of the cyclic reduction in log q steps: Each processor computes the first $Q = \log q$ steps of the cyclic reduction algorithm, i.e.,processor P_i computes for $k = 1, ..., Q$ the values

$$a_j^{(k)},\ b_j^{(k)},\ c_j^{(k)} \text{ and } y_j^{(k)}$$

for $j = (i-1) \cdot q + 2^k, ..., i \cdot q$ with step size 2^k.
After each computation step, processor P_i receives four data values from P_{i-1} (if $i > 1$) and from processor P_{i+1} (if $i < n$) computed in the previous step. Since each processor owns a block of rows of size q, no communication with any other processor is required. The size of data to be exchanged with the neighboring processors is a multiple of 4 since four coefficients $a_j^{(k)}$, $b_j^{(k)}$, $c_j^{(k)}$, and $y_j^{(k)}$ are transferred. Only one data block is received per step and so there are at most $2Q$ messages of size 4 for each step.

Cyclic Reduction for Tridiagonal Systems

- **Phase 2:** Parallel recursive doubling for tridiagonal systems of size p: Processor P_i is responsible for the ith equation of the following p-dimensional tridiagonal system

$$\bar{a}_i \bar{x}_{i-1} + \bar{b}_i \bar{x}_i + \bar{c}_i \bar{x}_{i+1} = \bar{y}_i, \text{ for } i = 1, ..., p$$

with

$$\left.\begin{array}{rcl} \bar{a}_i & = & a_{i.q}^{(Q)} \\ \bar{b}_i & = & b_{i.q}^{(Q)} \\ \bar{c}_i & = & c_{i.q}^{(Q)} \\ \bar{y}_i & = & y_{i.q}^{(Q)} \\ \bar{x}_i & = & x_{i.q}^{(Q)} \end{array}\right\} \text{ for } i =!, ..., p.$$

For the solution of this system, we use recursive doubling. Each processor is assigned one equation. Processor P_i performs $\lceil \log p \rceil$ steps of the recursive doubling algorithm.

Cyclic Reduction for Tridiagonal Systems

- In step k processor P_i computes

$$\bar{a}_i^{(k)},\ \bar{b}_i^{(k)},\ \bar{c}_i^{(k)} \text{ and } \bar{y}_i^{(k)}$$

for which the values of

$$\bar{a}_i^{(k-1)},\ \bar{b}_i^{(k-1)},\ \bar{c}_i^{(k-1)} \text{ and } \bar{y}_i^{(k-1)}$$

from the previous step computed by a different processor are required. Thus, there is a communication in each of the $\lceil \log p \rceil$ steps with a message size of four values. After step $N' = \lceil \log p \rceil$ processor P_i computes

$$\bar{x}_i = \bar{y}_i^{(N')} / \bar{b}_i^{(N')}.$$

- **Phase 3:** Parallel substitution of cyclic reduction: After the second phase, the values $\bar{x}_i = x_i \cdot q$ are already computed. In this phase, each processor P_i, $i = 1, ..., p$, computes the values x_j with $j = (i-1)q+1, ..., iq-1$ in several steps according to the substitution rule. In step k, $k = Q-1, ..., 0$, the elements x_j, $j = 2^k, ..., n$, with step size 2^{k+1} are computed.

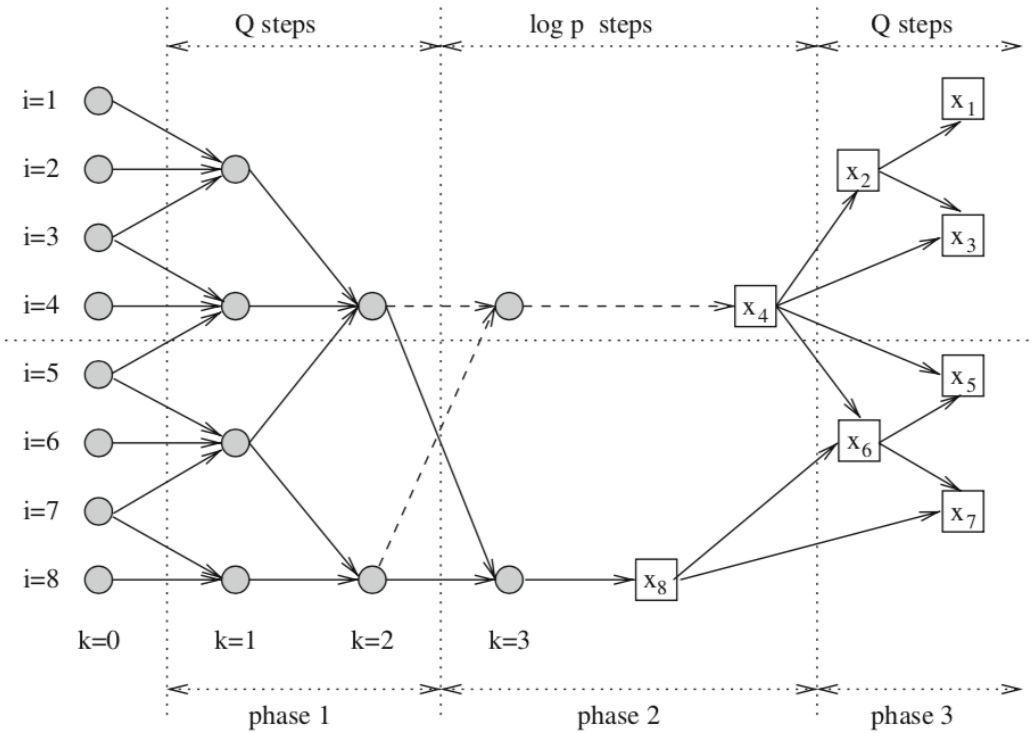


Illustration of the parallel algorithm for the cyclic reduction for $n = 8$ equations and $p = 2$ processors. Each of the processors is responsible for $q = 4$ equations; we have $Q = 2$. The first and the third phases of the computation have $\log q = 2$ steps. The second phase has $\log p = 1$ step. As recursive doubling is used in the second phase, there are more components of the solution to be computed in the second phase co

Cyclic Reduction: Parallel Execution Time

- The execution time of the parallel algorithm can be modeled by the following run- time functions. Phase 1 executes

$$Q = \log q = \log \frac{n}{p} = \log n - \log p$$

steps where in step k with $1 \leq k \leq Q$ each processor computes at most $q/2^k$ coefficient blocks of 4 values each.
- Each coefficient block requires 14 arithmetic operations. The computation time of phase 1 can therefore be estimated as

$$T_1(n,p) = 14t_{op} \cdot \sum_{k=1}^{Q} \frac{q}{2^k}$$

Moreover, each processor exchanges in each of the Q steps two messages of 4 values each with its two neighboring processors by participating in single transfer operations.
- Since in each step the transfer operations can be performed by all processors in parallel without interference, the resulting communication time is

$$C_1(n,p) = 2Q \cdot 4t_w = 2\log \frac{n}{p} \cdot 4t_w.$$

Cyclic Reduction: Parallel Execution Time

- Phase 2 executes $\lceil \log p \rceil$ steps. In each step, each processor computes 4 coefficients requiring 14 arithmetic operations. Then the value $\bar{x}_i = x_i \cdot q$ is computed by a single arithmetic operation. The computation time is therefore

$$T_2(n,p) = 14\lceil \log p \rceil \cdot 4t_{op} + t_{op}.$$

  In each step, each processor sends and receives 4 data values from other processors, leading to a communication time

$$C_2(n,p) = 2\lceil \log p \rceil \cdot 4t_w.$$

  In each step k of phase 3, $k = 0, ..., Q-1$, each processor computes 2^k components of the solution vector. For each component, five operations are needed. Altogether, each processor computes $\sum_{k=0}^{Q-1} 2^k = 2^Q - 1 = q - 1$ components with one component already computed in phase 2. The resulting computation time is

$$T_3(n,p) = 5(q-1) \cdot t_{op} = 5(n/p - 1) \cdot t_{op}.$$

Cyclic Reduction: Parallel Execution Time

- Moreover, each processor exchanges one data value with each of its neighboring processors; the communication time is therefore

$$C_3(n,p) = 2 \cdot t_w.$$

The resulting total computation time is

$$T(n,p) = \left(14\frac{n}{p} + 14 \cdot \lceil \log p \rceil + 5\frac{n}{p} - 4\right) \cdot t_{op} \approx \left(19\frac{n}{p} + 14 \cdot \log p\right) \cdot t_{op}.$$

The communication overhead is

$$C(n,p) = \left(2log\frac{n}{p} + 2 \cdot \lceil \log p \rceil\right) 4t_w + 2t_w \approx 2 \log n \cdot 4t_w + 2t_w.$$

- Compared to the sequential algorithm, the parallel implementation leads to a small computational redundancy of $14 \log p$ operations. The communication overhead increases logarithmically with the number of rows, whereas the computation time increases linearly.

Solving the Discretized Poisson Equation 1D

- The cyclic reduction algorithm for banded matrices is suitable for the solution of the discretized two-dimensional Poisson equation.
- The special structure has only four non-zero diagonals and the band has a sparse structure.
- The use of the Gaussian elimination method would not preserve the sparse banded structure of the matrix, since the forward elimination for eliminating the two lower diagonals leads to fill-ins with non-zero elements between the two upper diagonals. This induces a higher computational effort which is needed for banded matrices with a dense band of semi-bandwidth N .
- It is possible to build a method of cyclic reduction for banded matrices, which preserves the sparse banded structure.

Solving the Discretized Poisson Equation 1D

- We call the discretized Poisson equation $Az = d$ considering A blocked tridiagonal matrix as before.
- Using the notation for the banded system, we get

$$B_i^{(0)} := \frac{1}{h^2} B \text{ for } i = 1, ..., N,$$
$$A_i^{(0)} := \frac{1}{h^2} I \text{ and } C_i^{(0)} := \frac{1}{h^2} I \text{ for } i = 1, ..., N.$$

The vector $d \in \mathbb{R}^n$ consists of N subvectors $D_j \in \mathbb{R}^N$, i.e.,

$$d = \begin{pmatrix} D_1 \\ \vdots \\ D_N \end{pmatrix}, \quad \text{with } D_j = \begin{pmatrix} d_{(j-1)N+1} \\ \vdots \\ d_{jN} \end{pmatrix}.$$

Solving the Discretized Poisson Equation 1D

Analogously, the solution vector consists of N subvectors Z_j of length N each, i.e.,

$$z = \begin{pmatrix} Z_1 \\ \vdots \\ Z_N \end{pmatrix}, \quad \text{with } Z_j = \begin{pmatrix} z_{(j-1)N+1} \\ \vdots \\ z_{jN} \end{pmatrix}.$$

The initialization for the cyclic reduction algorithm is given by

$$\begin{aligned} B^{(0)} &:= B \\ D_j^{(0)} &:= D_j \text{ for } i = 1, ..., N, \\ D_j^{(k)} &:= 0 \text{ for } k = 0, ..., [\log N],\ j \in \mathbb{Z} \setminus \{1, ..., N\}, \\ Z_j &:= 0 \text{ for } j \in \mathbb{Z} \setminus \{1, ..., N\}. \end{aligned}$$

Solving the Discretized Poisson Equation 1D

In step k of the cyclic reduction, $k = 1, ..., [\log N]$, the matrices $B^{(k)} \in \mathbb{R}^{N \times N}$ and the vectors $D_j^{(k)} \in \mathbb{R}^N$ for $j = 1, ..., N$ are computed according to

$$B^{(k)} = (B^{(k-1)})^2 - 2I,$$
$$D_j^{(k)} = D_{j-2^{k-1}} + B^{(k-1)} D_j^{(k-1)} + D_{j+2^{k-1}}^{(k-1)}.$$

For $k = 0, ..., [\log N]$ the equation of a cycling reduction has the special form

$$-Z_{j-2^k} + B^{(k)} Z_j - Z_{j+2^k} = D_j^{(k)} \quad \text{for } j = 1, ..., n.$$

Solving the Discretized Poisson Equation 1D

These three equations represent the method of cyclic reduction for the discretized Poisson equation, which can be seen by induction. For $k = 0$, the inizialization provides the initial equation system $Az = d$. For $0 < k < [\log N]$ and $j \in \{1, ..., N\}$ the three equations

$$\begin{array}{lllll} -Z_{j-2^{k+1}} & +B^{(k)}_{j-2^k} & -Z_j & & = D_{j-2^k}, \\ & -Z_{j-2^k} & +B^{(k)}Z_j & -Z_{j+2^k} & = D^{(k)}_i, \\ & & -Z_j & +B^{(k)}Z_{j+2^k} & -x_{j+2^{k+1}} = D^{(k)}_{j+2^k}, \end{array}$$

are considered. Now if we multiply the general equation by $B^{(k)}$ from the left results in

$$-B^{(k)}Z_{j-2^k} + B^{(k)}B^{(k)}Z_j - B^{(k)}Z_{j+2^k} = B^{(k)}D^{(k)}_j.$$

Now adding this to the previous equation it gives the expected cyclic reduction structure

$$-Z_{j-2^{k+1}} - Z_j + B^{(k)}B^{(k)}Z_j - Z_j - Z_{j+2^{k+1}} = D_{j-2^k} + B^{(k)}D^{(k)}_j + D^{(k)}_{j+2^k},$$

Solving the Discretized Poisson Equation 1D

In summary, the cyclic reduction for the discretized two-dimensional Poisson equation consists of the following two steps:

1. Elimination phase: For $k = 1, ..., [logN]$, the matrices $B^{(k)}$ and the vectors $D_j^{(k)}$ are computed for $j = 2^k, ..., N$ with step size 2^k
2. Substitution phase: For $k = [\log N], ..., 0$, the linear equation system

$$B^{(k)} Z_j = D^{(k)} + Z_{j-2^k} + Z_{j+2^k}$$

for $j = 2^k, ..., N$ with step size 2^{k+1} is solved.

In the first phase, $[\log N]$ matrices and $O(N)$ subvectors are computed. The computation of each matrix includes a matrix multiplication with time $O(N^3)$. The computation of a subvector includes a matrix–vector multiplication with complexity $O(N^2)$. Thus, the first phase has a computational complexity of $O(N^3 \log N)$. In the second phase, $O(N)$ linear equation systems are solved. This requires time $O(N^3)$ when the special structure of the matrices $B^{(k)}$ is not exploited.

Solving the Discretized Poisson Equation 2D

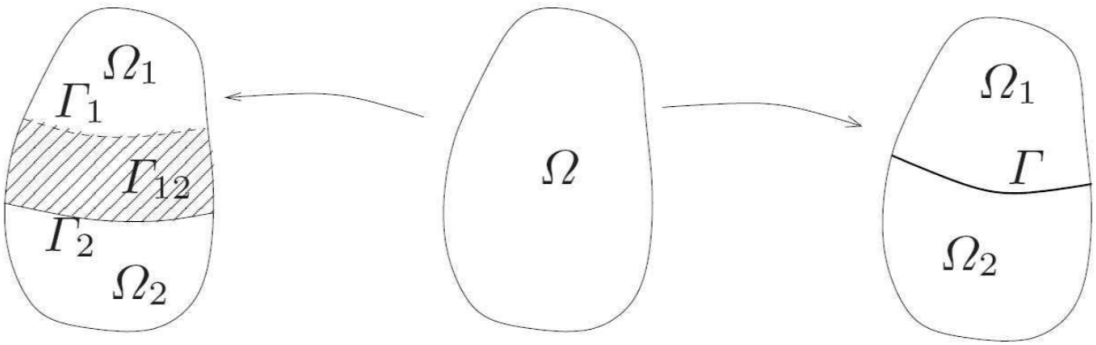
- In the case of a 2D Poisson equation, the banded structure of the matrix A is replaced by a block-banded structure.
- This requires an additional work where all the blocks are managed using a cyclic reduction technique.
- An easy and fast to implement alternative is represented by Domain Decomposition.

Poisson Equation 2D: Domain Decomposition Methods

- Domain Decomposition can be used in the framework of any discretization method for PDEs (FEM, FV, FD, SEM) to make their algebraic solution more efficient on parallel computer platforms
- Domain Decomposition Methods allow the reformulation of a boundary-value problem on a partition of the computational domain into subdomains
- very convenient framework for the solution of heterogeneous or multiphysics problems, i.e. those that are governed by differential equations of different kinds in different subregions of the computational domain.

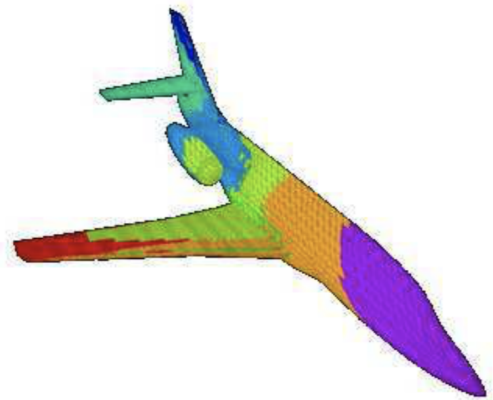
Domain Decomposition Methods: Basic Idea

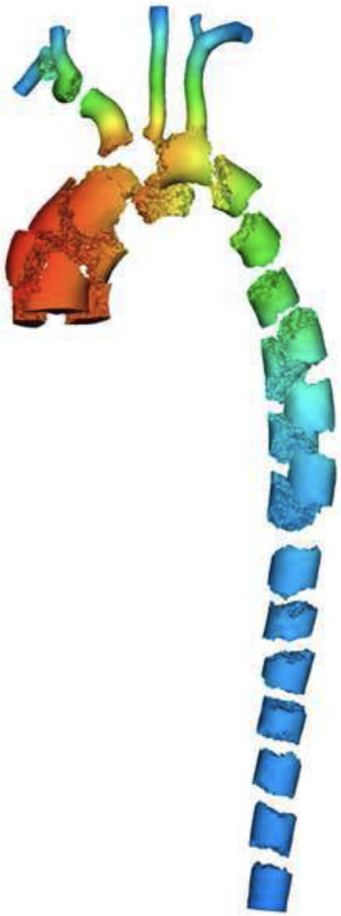
- By Domain Decomposition Methods methods the computational domain where the boundary value problem is set is subdivided into two or more subdomains on which discretized problems of smaller dimension are to be solved.
- Parallel solution algorithms may be used.
- There are two ways of subdividing the computational domain: with disjoint subdomains and with overlapping subdomains



Correspondingly, different DD algorithms will be set up.

Examples of Subdivisions in Applications





Domain Decomposition Methods: Model Problem

Consider the model problem: find $u : \Omega \to \mathbb{R}$ s.t.

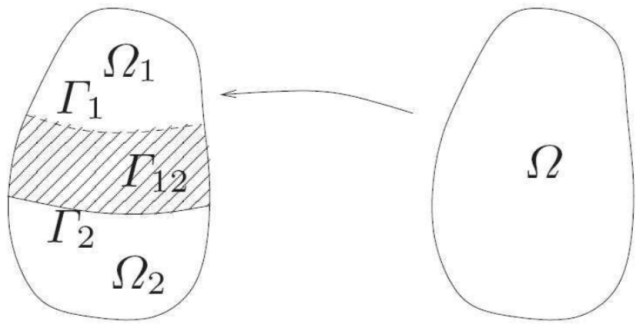
$$\begin{cases} Lu = f & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases}$$

L is a generic second order elliptic operator.

Example: $Lu = \Delta u$ (Poisson Equation)

Schwarz Methods

Consider a decomposition of Ω with overlap:



The iterative method: given $u_2^{(0)}$ on Γ_1, for $k \geq 1$:

$$\text{solve} \quad \begin{cases} Lu_1^{(k)} = f & \text{in } \Omega_1 \\ u_1^{(k)} = u_2^{(k-1)} & \text{on } \Gamma_1 \\ u_1^{(k)} = 0 & \text{on } \partial\Omega_1 \setminus \Gamma_1 \end{cases}$$

$$\text{solve} \quad \begin{cases} Lu_2^{(k)} = f & \text{in } \Omega_2 \\ u_2^{(k)} = \begin{cases} u_1^{(k)} \\ u_1^{(k-1)} \end{cases} & \text{on } \Gamma_2 \\ u_2^{(k)} = 0 & \text{on } \partial\Omega_2 \setminus \Gamma_2 \, . \end{cases}$$

Choice of the trace on Γ_2:

- if $u_1^{(k)} \Rightarrow$ *multiplicative Schwarz method*
- if $u_1^{(k-1)} \Rightarrow$ *additive Schwarz method*

We have two elliptic bvp with Dirichlet conditions in Ω_1 and Ω_2, and we wish the two sequences $\{u_1^{(k)}\}$ and $\{u_2^{(k)}\}$ to converge to the restrictions of the solution u of the original problem:

$$\lim_{k\to\infty} u_1^{(k)} = u_{|\Omega_1} \text{ and } \quad \lim_{k\to\infty} u_2^{(k)} = u_{|\Omega_2} \ .$$

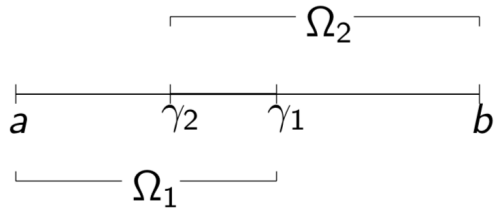
The Schwarz method applied to the model problem always converges, with a rate that increases as the measure $|\Gamma_{12}|$ of the overlapping region Γ_{12} increases.

One Example

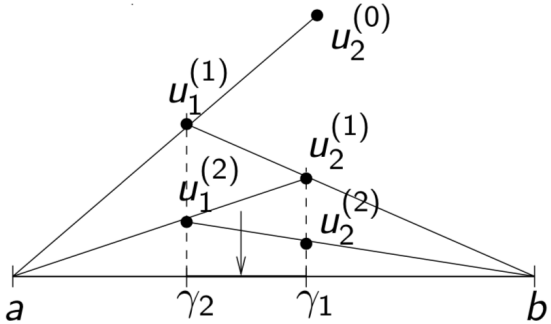
Example. Consider the model problem

$$\begin{cases} -u''(x) = 0 & a < x < b, \\ u(a) = u(b) = 0, \end{cases}$$

where



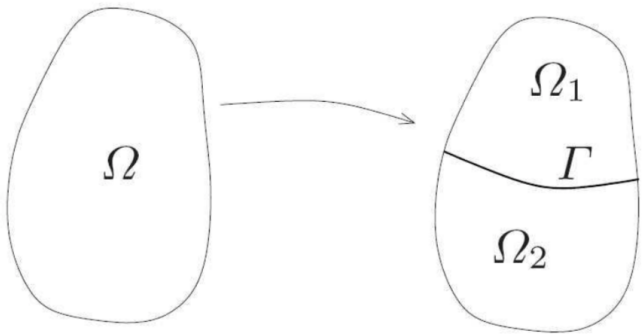
The solution is $u = 0$. We show a few iterations of the method:



Clearly, the method converges with a rate that reduces as the length of the interval (γ_2, γ_1) gets smaller.

Non-Overlapping Decomposition

We partition now the domain Ω in two disjoint subdomains:



The following equivalence result holds.

Theorem
The solution u of the model problem is such that $u_{|\Omega_i} = u_i$ for $i = 1, 2$, where u_i is the solution to the problem

$$\begin{cases} Lu_i = f & \text{in } \Omega_i \\ u_i = 0 & \text{on } \partial\Omega_i \setminus \Gamma \end{cases}$$

with interface conditions

$$u_1 = u_2 \qquad \text{and} \qquad \frac{\partial u_1}{\partial n} = \frac{\partial u_2}{\partial n} \qquad \text{on } \Gamma.$$

Dirichlet-Neumann Method

Given $u_2^{(0)}$ on Γ, for $k \geq 1$ solve the problems:

$$\begin{cases} Lu_1^{(k)} = f & \text{in } \Omega_1 \\ u_1^{(k)} = u_2^{(k-1)} & \text{on } \Gamma \\ u_1^{(k)} = 0 & \text{on } \partial\Omega_1 \setminus \Gamma \,, \end{cases}$$

$$\begin{cases} Lu_2^{(k)} = f & \text{in } \Omega_2 \\ \dfrac{\partial u_2^{(k)}}{\partial n} = \dfrac{\partial u_1^{(k)}}{\partial n} & \text{on } \Gamma \\ u_2^{(k)} = 0 & \text{on } \partial\Omega_2 \setminus \Gamma \,. \end{cases}$$

The equivalence theorem guarantees that when the two sequences $\{u_1^{(k)}\}$ and $\{u_2^{(k)}\}$ converge, then their limit will be necessarily the solution to the exact problem.
The DN algorithm is therefore consistent.
Its convergence however is not always guaranteed.

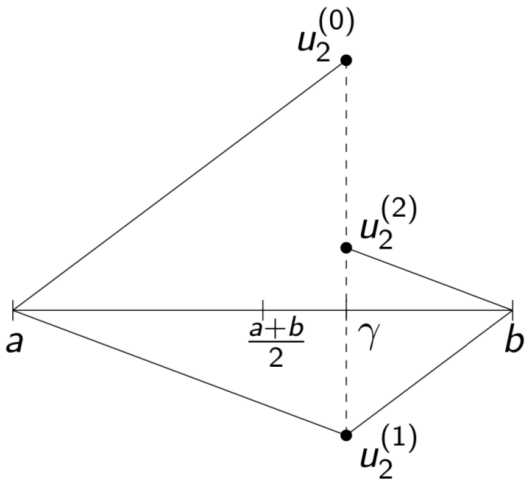
Example:

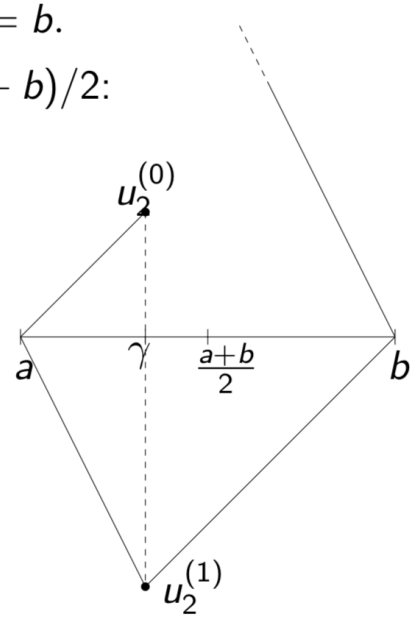
Example. Let $\Omega = (a, b)$, $\gamma \in (a, b)$, $L = -d^2/dx^2$ and $f = 0$. At every $k \geq 1$ the DN algorithm generates the two subproblems:

$$\begin{cases} -(u_1^{(k)})'' = 0 & a < x < \gamma \\ u_1^{(k)} = u_2^{(k-1)} & x = \gamma \\ u_1^{(k)} = 0 & x = a \end{cases}$$

$$\begin{cases} -(u_2^{(k)})'' = 0 & \gamma < x < b \\ (u_2^{(k)})' = (u_1^{(k)})' & x = \gamma \\ u_2^{(k)} = 0 & x = b. \end{cases}$$

The two sequences converge only if $\gamma > (a + b)/2$:





A variant of the DN algorithm can be set up by replacing the Dirichlet condition in the first subdomain by

$$u_1^{(k)} = \theta u_2^{(k-1)} + (1-\theta) u_1^{(k-1)} \quad \text{on } \Gamma \ ,$$

that is by introducing a *relaxation* which depends on a positive parameter θ.
In such a way it is always possible to reduce the error between two subsequent iterates.

In the previous example, we can easily verify that, by choosing

$$\theta_{opt} = -\frac{u_1^{(k-1)}}{u_2^{(k-1)} - u_1^{(k-1)}} \ ,$$

the algorithm converges to the exact solution in a single iteration.

More in general, there exists a suitable value $\theta_{max} < 1$ such that the DN algorithm converges for any possible choice of the relaxation parameter θ in the interval $(0, \theta_{max})$.

Neumann Neumann Method

Consider again a partition of Ω into two disjoint subdomains and denote by λ the (unknown) value of the solution u at their interface Γ.

Consider the following iterative algorithm: for any given $\lambda^{(0)}$ on Γ, for $k \geq 0$ and $i = 1, 2$, solve the following problems:

$$\begin{cases} -\triangle u_i^{(k+1)} = f & \text{in } \Omega_i \\ u_i^{(k+1)} = \lambda^{(k)} & \text{on } \Gamma \\ u_i^{(k+1)} = 0 & \text{on } \partial\Omega_i \setminus \Gamma \,, \end{cases}$$

$$\begin{cases} -\triangle \psi_i^{(k+1)} = 0 & \text{in } \Omega_i \\ \dfrac{\partial \psi_i^{(k+1)}}{\partial n} = \dfrac{\partial u_1^{(k+1)}}{\partial n} - \dfrac{\partial u_2^{(k+1)}}{\partial n} & \text{on } \Gamma \\ \psi_i^{(k+1)} = 0 & \text{on } \partial\Omega_i \setminus \Gamma \,, \end{cases}$$

with

$$\lambda^{(k+1)} = \lambda^{(k)} - \theta \left(\sigma_1 \psi_{1|\Gamma}^{(k+1)} - \sigma_2 \psi_{2|\Gamma}^{(k+1)} \right),$$

where θ is a positive acceleration parameter, while σ_1 and σ_2 are two positive coefficients.

Parallel Numerical Solution of the 2D Heat Equation

We consider developing a parallel numerical solver for the 2D heat equation

$$\partial_t u = \Delta u$$

on the domain $\Omega = [0, 1]^2$ (i.e., the unit square) with the initial conditions

$$u(x, y, 0) = f(x, y)$$

and the boundary conditions

$$u(0, y, t) = u(1, y, t) = u(x, 0, t) = u(x, 1, t) = 0.$$

We begin by dividing each of the two spatial dimensions into N intervals of uniform size. In other words, we have $Nh = 1$, where h is the step size in space.

Parallel Numerical Solution of the 2D Heat Equation

We also discretize the time dimension with a uniform step size k, which we choose as

$$k = \frac{h^2}{4}$$

in order for the finite difference scheme to be stable. The discrete points on the grid are denoted (as usual) by

$$x_i = ih, \quad y_j = jh, \quad t_n = nk.$$

By restricting the function u to the grid, we obtain a discretization of u:

$$u_{i,j}^n = u(x_i, y_j, t_n).$$

Parallel Numerical Solution of the 2D Heat Equation

Let $v_{i,j}^n$ denote our approximation of $u_{i,j}^n$. If we use a finite difference scheme with explicit Euler as the time-stepping method, then we eventually arrive at the following formula for $v_{i,j}^{k+1}$:

$$v_{i,j}^{n+1} = v_{i,j}^n + \frac{k}{h^2}(v_{i-1,j}^n + v_{i,j-1}^n - 4v_{i,j}^n v_{i+1,j}^n + v_{i,j+1}^n)$$

The key point is that $v_{i,j}^{k+1}$ depends exclusively on the value of v^k at the point (x_i, y_j) as well as each of the four neighbouring points $(x_i \pm h, y_j \pm h)$. The update formula above is also known as a 5-point stencil.

Distributed-memory parallelization

- We distribute the grid points onto a $p_x \times p_y$ process mesh with a 2D block distribution.
- In order for the process that owns the central block to advance its points one time step, it must know the values of the neighbouring grid points:
- Note that these are stored on the four immediate neighbours of the process. Algorithm 1 outlines a straightforward parallel algorithm.

Algorithm 1

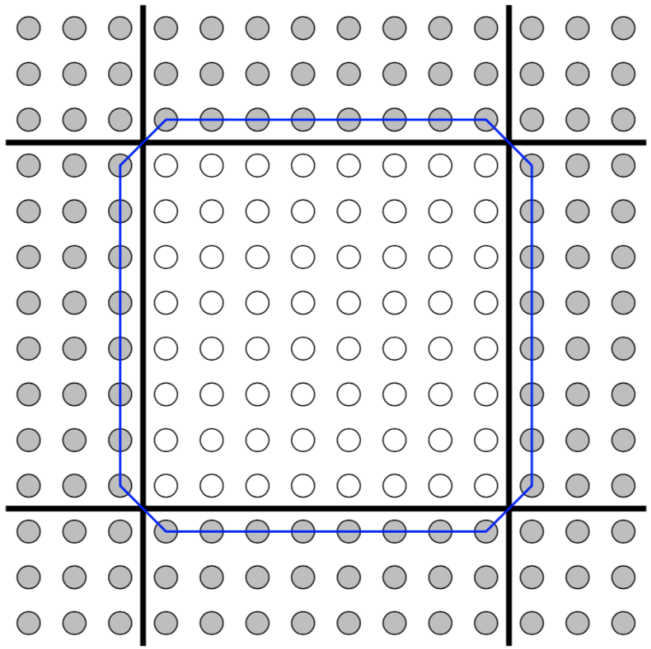


Figure 1: Illustration of a block of grid points and one level of neighbouring grid points.

Algorithm 1 Basic

1: **for** $k = 0, 1, \ldots$ **do**
2: Send the interior borders to the neighbouring processes.
3: Receive the exterior borders from the neighbouring processes.
4: Advance the local block one time step.
5: **end for**

Distributed-memory parallelization

- One drawback with Algorithm 1 is that the communications on line 2–3 are not overlapped with computations. Therefore, the parallel execution time will include the full cost of the communication. It is possible to overlap the communication with computation by observing that the interior grid points in the local block can be computed before the grid points on the border, which means that we can communicate the border while advancing the interior grid points. The resulting algorithm is outlined in Algorithm 2.
- Another drawback with Algorithm 1 is that we perform only $\theta(n^2/p)$ computations between each pair of communication phases. It is possible to increase the granularity of the computations by a factor of r by performing some redundant computations. The idea is to send in each communication phase all the points that we need to advance the local block r time steps.

Algorithm 2

Algorithm 2 Overlapped

1: **for** $k = 0, 1, \ldots$ **do**
2: Asynchronously send the interior borders to the neighbouring processes.
3: Asynchronously receive the exterior borders from the neighbouring processes.
4: Advance the *interior* of the local block one time step.
5: Wait until the sends and receives complete.
6: Advance the *borders* of the local block one time step.
7: **end for**

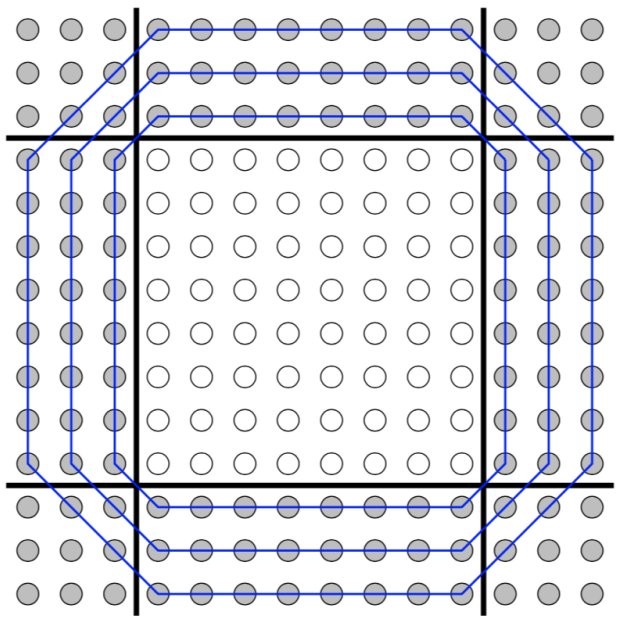


Figure 2: Illustration of a block of grid points and three levels of neighbouring grid points.

Distributed-memory parallelization

- The three blue loops in Figure 2 illustrate the points needed for the case $r = 3$. Note three things. First, we communicate with eight instead of four neighbouring processes. Second, we communicate slightly more grid points. Third, we perform redundant computations to advance the exterior borders locally. Algorithm 3 outlines the communication-avoiding Basic algorithm.
- Of course, we can also apply the same technique to increase the granularity in Algorithm 2. The resulting communication-avoiding Overlapped algorithm is outlined in Algorithm 4. Let us briefly touch upon the issue of appropriate data structures. A good option is to store the local $m_y \times m_x$ block at the center of a matrix of size $(m_y + 2r) \times (m_x + 2r)$. Then we can store the exterior borders received from the neighbouring processes in the extra space created around the local block in the center of the matrix.

Algorithm 3

Algorithm 3 Communication-Avoiding Basic

1: **for** $k = 0, r, \ldots$ **do**
2: Send the r interior borders to the neighbouring processes.
3: Receive the r exterior borders from the neighbouring processes.
4: Advance the local block r time steps.
5: **end for**

Algorithm 4 Communication-Avoiding Overlapped

1: **for** $k = 0, r, \ldots$ **do**
2: Asynchronously send the r interior borders to the neighbouring processes.
3: Asynchronously receive the r exterior borders from the neighbouring processes.
4: Advance the *interior* of the local block r time steps.
5: Wait until the sends and receives complete.
6: Advance the r *borders* of the local block r time steps.
7: **end for**