Introduction to parallel computing
part II

Aslak Tveito

University of Oslo

Geilo, March 2001

Partial Differential Equations

- $-u'' = f$
- $-u_{xx} - u_{yy} = f$
- $u_t = u_{xx}$
- $u_t = u_{xx} + u_{yy}$
- $u_t = u_{xx} + u_{yy} + f(u)$
- $u_t + f(u)_x + g(u)_y = 0$
- $-\nabla \cdot (k(u)\nabla u) = f$
- All PDE solvers consist of basic steps like
  - $\alpha = (x, y)$ (inner products)
  - $x = Ay$ (matrix-vector products)
  - $u^{n+1} = Au^n + F(u^n)$ (Simple iterations)
  - Communication of scalars or boundary values

Consider

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

Finite-difference approximation:

$$\begin{aligned} x_j = jh,\ h &= \frac{1}{N+1},\ j = 0,1,\ldots,N+1 \\ u_j &\approx u(x_j) \\ u_0 &= u_{N+1} = 0 \end{aligned}$$

Scheme:

$$-\frac{u_{j-1} - 2u_j + u_{j+1}}{h^2} = f(x_j),\ j = 1,2,\ldots N.$$

This results in the linear system

$$Au = b$$

where

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

and

$$u = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix}, \qquad b = \begin{pmatrix} -h^2 f(x_1) \\ -h^2 f(x_2) \\ \vdots \\ -h^2 f(x_n) \end{pmatrix}.$$

Methods for solving $Au = b$

- Gaussian elimination
- Jacobi's method
- Gauss-Seidels iteration
- The Conjugated Gradient Method
- Multigrid/Domain decomposition (covered in a seperate lecture)

Gaussian elimination for tridiagonal linear systems

We consider a system of the form

$$Au = b,$$

where the coefficient matrix A has the form

$$A = \begin{pmatrix} \alpha_1 & \gamma_1 & 0 & \dots & 0 \\ \beta_2 & \alpha_2 & \gamma_2 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \beta_{n-1} & \alpha_{n-1} & \gamma_{n-1} \\ 0 & \dots & 0 & \beta_n & \alpha_n \end{pmatrix}.$$

This system can be equivalently written in component form, i.e.

$$\begin{aligned}
\alpha_1 u_1 + \gamma_1 u_2 &= b_1, \\
\beta_2 u_1 + \alpha_2 u_2 + \gamma_2 u_3 &= b_2, \\
\beta_3 u_2 + \alpha_3 u_3 + \gamma_3 u_4 &= b_3, \\
\ddots \qquad &\vdots \\
\beta_{n-1} u_{n-2} + \alpha_{n-1} u_{n-1} + \gamma_{n-1} u_n &= b_{n-1}, \\
\beta_n u_{n-1} + \alpha_n u_n &= b_n.
\end{aligned}$$

Algorithm

$$
\begin{aligned}
&\delta_1 = \alpha_1 \\
&c_1 = b_1 \\
&\quad \text{for} \quad k = 2, 3, \ldots, n \\
&\qquad m_k = \beta_k / \delta_{k-1} \\
&\qquad \delta_k = \alpha_k - m_k \gamma_{k-1} \\
&\qquad c_k = b_k - m_k c_{k-1} \\
&v_n = c_n / \delta_n \\
&\quad \text{for} \quad k = n-1, n-2, \ldots, 1 \\
&\qquad v_k = (c_k - \gamma_k v_{k+1}) / \delta_k
\end{aligned}
$$

Sequential algorithm.

Jacobi's method

$$-\frac{u_{j-1} - 2u_j + u_{j+1}}{h^2} = f(x_j),\ j = 1, \ldots, N$$

$$\Rightarrow u_j = \frac{1}{2}\left(u_{j-1} + u_{j+1}\right) + \frac{h^2}{2} f(x_j),\ j = 1, \ldots, N$$

Iteration:

$$u_j^{k+1} = \frac{1}{2}\left(u_{j-1}^k + u_{j+1}^k\right) + \frac{h^2}{2} f(x_j),\ j = 1, \ldots, N$$

$$
\begin{aligned}
u_0^{k+1} &= 0 \\
u_1^{k+1} &= \frac{1}{2}\left(u_0^k + u_2^k\right) + \frac{h^2}{2} f(x_1) \\
u_2^{k+1} &= \frac{1}{2}\left(u_1^k + u_3^k\right) + \frac{h^2}{2} f(x_2) \\
&\vdots \\
u_N^{k+1} &= \frac{1}{2}\left(u_{N-1}^k + u_{N+1}^k\right) + \frac{h^2}{2} f(x_N) \\
u_{N+1}^{k+1} &= 0
\end{aligned}
$$

Easy to parallelize.

Parallel implementation

Let p be the number of processors and assume that

$$N = np$$

where $n \geq 1$ is an integer. Let u^0 be an initial guess ($u^0 = 0$ is ok).

Step 1

Distribute u^0 as follows:

$$\begin{pmatrix} u_0^0 \\ u_1^0 \\ \vdots \\ u_n^0 \\ u_{n+1}^0 \end{pmatrix} \text{ at proc. } 1, \quad \begin{pmatrix} u_n^0 \\ u_{n+1}^0 \\ \vdots \\ u_{2n}^0 \\ u_{2n+1}^0 \end{pmatrix} \text{ at proc. } 2,$$

$$\cdots \begin{pmatrix} u_{np-n}^0 \\ u_{np-n+1}^0 \\ \vdots \\ u_{np}^0 \end{pmatrix} \text{ at proc. } p.$$

Note that all “boundary” values are stored twice.

Step 2

Use the usual local numbering such that $u_0^0, u_1^0, \ldots, u_{n+1}^0$ are local variables at each processor.

Step 3

Given k, compute

$$u_j^{k+1} = \frac{1}{2}\left(u_{j-1}^k + u_{j+1}^k\right) + \frac{h^2}{2} f(x_j),\ j = 1, 2, \ldots, n$$

at each processor.

Step 4

Send/receive boundary values.

Summary

1. Initially: distribute data
2. For $k = 1, 2, \ldots$ convergence
   - On each processor: Compute u^{k+1} based on u^k
   - Exchange boundary values
3. (Gather)

Gauss-Seidels iteration

$$-\frac{u_{j-1} - 2u_j + u_{j+1}}{h^2} = f(x_j), j = 1, \ldots, N$$

$$\Rightarrow u_j = \frac{1}{2}(u_{j-1} + u_{j+1}) + \frac{h^2}{2} f(x_j), j = 1, \ldots, N$$

Iteration:

$$u_j^{k+1} = \frac{1}{2}\left(u_{j-1}^{k+1} + u_{j+1}^{k}\right) + \frac{h^2}{2} f(x_j), j = 1, \ldots, N$$

$$
\begin{aligned}
u_0^{k+1} &= 0 \\
u_1^{k+1} &= \frac{1}{2}\left(u_0^{k+1} + u_2^k\right) + \frac{h^2}{2} f(x_1) \\
u_2^{k+1} &= \frac{1}{2}\left(u_1^{k+1} + u_3^k\right) + \frac{h^2}{2} f(x_2) \\
&\vdots \\
u_N^{k+1} &= \frac{1}{2}\left(u_{N-1}^{k+1} + u_{N+1}^k\right) + \frac{h^2}{2} f(x_N) \\
u_{N+1}^{k+1} &= 0
\end{aligned}
$$

Difficult to parallelize.

The CG Algorithm

<u>declare:</u>

k: integer;

ξ, β, $\pi^0_{r,r}$, $\pi^1_{r,r}$, $\pi_{p,w}$: real;

p, r, w, x: vector;

<u>assign:</u>

$r = b - Ax$;

$p = r$;

$\pi^0_{r,r} = (r, r)$;

for k: $0 \leq k < K$ ::

$w = Ap$; // (1) matrix-vector product

$\pi_{p,w} = (p, w)$; // (2) inner product

$\xi = \pi^0_{r,r} / \pi_{p,w}$; // (3)

$x = x + \xi p$; // (4) vector+scalar*vector

$r = r - \xi w$; // (5) vector+scalar*vector

$\pi^1_{r,r} = (r, r)$; // (6) inner product

$\beta = \pi^1_{r,r} / \pi^0_{r,r}$; // (7)

$p = r + \beta p$; // (8) vector+scalar*vector

$\pi^0_{r,r} = \pi^1_{r,r}$ // (9)

end

<u>end</u>

Essential operations

a) Matrix-vector-operation (1)

b) inner products (2), (6)

c) vector + scalar$*$vector (4), (5), (8)

We have considered b) and c); note that c) is basically the same as adding two vectors.

Let us consider the matrix-vector product.

Matrix-vector product

We want to solve

$$Au = b$$

using the CG method. Recall that A is given by

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

Let z be a <u>given</u> vector. Then we want to compute

$$w = Az$$

cf. the CG method.

We assume that z has components $z_0, z_1, \ldots, z_N$ with $z_0 = z_{N+1} = 0$. Then

$$w_j = -z_{j-1} + 2z_j - z_{j+1}, j = 1, \ldots, N$$

and $w_0 = w_{N+1} = 0$.

Parallel implementation

We store the vectors as we did in the case of the Jacobi-method.

Step 1

At each processor, compute:

$$w_j = -z_{j-1} + 2z_j - z_{j+1},\ j = 1, \ldots, n$$

using local variables.

Step 2

Interchange boundary values (for the next CG-iteration).

The CG Algorithm

declare:
    k: integer;
    ξ, β, $\pi^0_{r,r}$, $\pi^1_{r,r}$, $\pi_{p,w}$: real;
    p, r, w, x: vector;
assign:
    $r = b - Ax$;
    $p = r$;
    $\pi^0_{r,r} = (r, r)$;
    for k: $0 \leq k < K$ ::
        $w = Ap$; // (1) matrix-vector product
        $\pi_{p,w} = (p, w)$; // (2) inner product
        $\xi = \pi^0_{r,r}/\pi_{p,w}$; // (3)
        $x = x + \xi p$; // (4) vector+scalar*vector
        $r = r - \xi w$; // (5) vector+scalar*vector
        $\pi^1_{r,r} = (r, r)$; // (6) inner product
        $\beta = \pi^1_{r,r}/\pi^0_{r,r}$; // (7)
        $p = r + \beta p$; // (8) vector+scalar*vector
        $\pi^0_{r,r} = \pi^1_{r,r}$ // (9)
    end
end

The Poisson Equation, 2D

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

$$x_i = ih,\ h = \frac{1}{N+1},\ i = 0, 1, \ldots, N$$

$$y_j = jh,\ h = \frac{1}{N+1},\ j = 0, 1, \ldots, N$$

$$u_{ij} \approx u(x_i, y_j)$$

$$u_{ij} = 0 \text{ for all } i = 0,\ j = 0,\ i = N+1 \text{ and } j = N+1.$$

Scheme:

$$\begin{aligned} -\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} &- \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h^2} \\ &= f(x_i, y_j) \\ i, j &= 1, 2, \ldots, N \end{aligned}$$

This is a system of N^2 equations and N^2 unknowns.

Jacobi's method

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

$$\Rightarrow u_{i,j} = \frac{1}{4}\left(u_{i-1,j} + u_{i,j-1} + u_{i+1,j} + u_{i,j+1}\right) + \frac{h^2}{4} f_{i,j}$$

Iteration:

$$u_{i,j}^{k+1} = \frac{1}{4}\left(u_{i-1,j}^{k} + u_{i,j-1}^{k} + u_{i+1,j}^{k} + u_{i,j+1}^{k}\right) + \frac{h^2}{4} f_{i,j}$$

Implementation

proc 7	proc 8	proc 9
proc 4	proc 5	proc 6
proc 1	proc 2	proc 3

Step 1: Distribute data

Step 2:

- Update u on each proc. according to the scheme

$$u_{i,j}^{k+1} = \frac{1}{4}\left(u_{i-1,j}^{k} + u_{i,j-1}^{k} + u_{i+1,j}^{k} + u_{i,j+1}^{k}\right) + \frac{h^2}{4} f_{i,j}$$

- Interchange boundary values

The CG Algorithm

<u>declare:</u>

k: integer;

ξ, β, $\pi^0_{r,r}$, $\pi^1_{r,r}$, $\pi_{p,w}$: real;

p, r, w, x: vector;

<u>assign:</u>

$r = b - Ax$;

$p = r$;

$\pi^0_{r,r} = (r, r)$;

for k: $0 \leq k < K$ ::

$w = Ap$;	// (1) matrix-vector product
$\pi_{p,w} = (p, w)$;	// (2) inner product
$\xi = \pi^0_{r,r}/\pi_{p,w}$;	// (3)
$x = x + \xi p$;	// (4) vector+scalar*vector
$r = r - \xi w$;	// (5) vector+scalar*vector
$\pi^1_{r,r} = (r, r)$;	// (6) inner product
$\beta = \pi^1_{r,r}/\pi^0_{r,r}$;	// (7)
$p = r + \beta p$;	// (8) vector+scalar*vector
$\pi^0_{r,r} = \pi^1_{r,r}$	// (9)

end

<u>end</u>

Matrix-vector multiplication

$$w = Az$$

$$w_{i,j} = -z_{i-1,j} - z_{i,j-1} - z_{i+1,j} - z_{i,j+1} + 4z_{i,j} \qquad (4)$$

Step 1: Compute (4).

Step 2: Interchange boundary values.

The heat equation

$$
\begin{aligned}
u_t &= u_{xx},\ x \in (0,1) \\
u(x,0) &= f(x) \\
u(0,t) &= u(1,t) = 0,\ t \geq 0
\end{aligned}
$$

Finite difference scheme:

$$
\begin{aligned}
x_j = jh,\ h &= \frac{1}{N+1},\ j = 0, 1, \ldots, N+1 \\
t_k &= k\Delta t,\ \Delta t > 0 \\
u_j^k &\approx u(x_j, t_k)
\end{aligned}
$$

Explicit scheme:

$$
\frac{u_j^{k+1} - u_j^k}{\Delta t} = \frac{u_{j-1}^k - 2u_j^k + u_{j+1}^k}{h^2} \quad , \quad j = 1, \ldots, N,\ k \geq 0
$$

$$
u_0^k = u_{N+1}^k = 0,\ k \geq 0.
$$

$$
u_j^{k+1} = \alpha u_{j-1}^k + (1 - 2\alpha) u_j^k + \alpha u_{j+1}^k \tag{5}
$$

where $\alpha = \Delta t / h^2$.

Note that $\alpha \leq 1/2$ for stability.

Parallel implementation

Step 1: Distribute u^0 to p processors.

Step 2: At each processor:

For $k = 0, 1, 2, \ldots,$

1. Update u^{k+1} according to (5).
2. Interchange boundary values.

Implicit scheme

$$\frac{u_j^k - u_j^{k-1}}{\Delta t} = \frac{u_{j-1}^k - 2u_j^k + u_{j+1}^k}{h^2}$$

$$\Rightarrow -\alpha u_{j-1}^k + (1+2\alpha)u_j^k - \alpha u_{j+1}^k = u_j^{k-1},\ j = 1, \ldots, N$$

where $\alpha = \Delta t/h^2$.

This is a linear system on the form

$$Au^k = u^{k-1}$$

and can be solved as the similar problem arising from discretizing $-u'' = f$.

Heat equation 2D

$$
\begin{aligned}
u_t &= u_{xx} + u_{yy},\ (x,y) \in [0,1] \times [0,1] \\
u(x,y,0) &= f(x,y) \\
u &= 0 \text{ at } \partial\Omega
\end{aligned}
$$

Explicit scheme

$$
\begin{aligned}
\frac{u_{i,j}^{k+1} - u_{i,j}^{k}}{\Delta t} &= \frac{u_{i-1,j}^{k} - 2u_{i,j}^{k} + u_{i+1,j}^{k}}{h^2} \\
&+ \frac{u_{i,j-1}^{k} - 2u_{i,j}^{k} + u_{i,j+1}^{k}}{h^2}
\end{aligned}
$$

$$
u_{i,j}^{k+1} = \alpha \left(u_{i-1,j}^{k} + u_{i+1,j}^{k} + u_{i,j-1}^{k} + u_{i,j+1}^{k} \right) + (1 - 4\alpha) u_{i,j}^{k} \tag{6}
$$

Step 1: Distribute $u^0 = f$ to the p processors

Step 2: At each processor: For $k = 0, 1, \ldots$

- Update u^{k+1} according to (6)
- Interchange boundary values.

Implicit scheme

$$\frac{u_{i,j}^k - u_{i,j}^{k-1}}{\Delta t} = \frac{u_{i-1,j}^k - 2u_{i,j}^k + u_{i+1,j}^k}{h^2} + \frac{u_{i,j-1}^k - 2u_{i,j}^k + u_{i,j+1}^k}{h^2}$$

$$-\alpha\left(u_{i-1,j}^k + u_{i+1,j}^k + u_{i,j-1}^k + u_{i,j+1}^k\right) + (1+4\alpha)u_{i,j}^k = u_{i,j}^{k-1}$$

$$Au^k = u^{k-1}$$

Can be solved as the discrete version of $-u_{xx} - u_{yy} = f$.

Heat equation with nonlinear reaction

$$u_t = u_{xx} + u_{yy} + f(u)$$

Explicit scheme:

$$\begin{aligned} u_{i,j}^{k+1} &= \alpha\left(u_{i-1,j}^k + u_{i+1,j}^k + u_{i,j-1}^k + u_{i,j+1}^k\right) \\ &+ (1-4\alpha)u_{i,j}^k + \Delta t f(u_{i,j}^k) \end{aligned}$$

Implementation as for the linear case.

Implicit scheme

$$u_t = u_{xx} + f(u)$$

$$\frac{u_j^k - u_j^{k-1}}{\Delta t} = \frac{u_{j-1}^k - 2u_j^k + u_{j+1}^k}{h^2} + f(u_j^k)$$

$$\begin{aligned} -\alpha u_{j-1}^k + (1+2\alpha)u_j^k \ &- \ \alpha u_{j-1}^k - \Delta t f(u_j^k) = u_j^{k-1} \\ j \ &= \ 1, \ldots, N \end{aligned}$$

Non-linear system.

Newtons method $\Rightarrow$ a linear system $Au = b$ has to be solved repeatedly within each time-step.

Similar for $u_t = u_{xx} + u_{yy} + f(u)$.

Hyperbolic conservation laws

$$\begin{aligned} u_t + f(u)_x + g(u)_y &= 0 \\ u(x,y,0) &= u^0(x,y) \end{aligned}$$

Lax-Friedrichs:

$$\frac{u_{i,j}^{k+1} - \frac{1}{4}\left(u_{i-1,j}^k + u_{i+1,j}^k + u_{i,j-1}^k + u_{i,j+1}^k\right)}{\Delta t}$$
$$+\frac{f(u_{i+1,j}^k) - f(u_{i-1,j}^k)}{2h} + \frac{g(u_{i,j+1}^k) - g(u_{i,j-1}^k)}{2h} = 0$$

$$\begin{aligned} u_{i,j}^{k+1} &= \frac{1}{4}\left(u_{i-1,j}^k + u_{i+1,j}^k + u_{i,j-1}^k + u_{i,j+1}^k\right) \\ &- \frac{\Delta t}{2h}\left(f(u_{i+1,j}^k) - f(u_{i-1,j}^k)\right) \\ &- \frac{\Delta t}{2h}\left(g(u_{i,j+1}^k) - g(u_{i,j-1}^k)\right) \end{aligned} \tag{7}$$

Step 1: Distribute u^0 to the p processors.

Step 2: At each processor:

- Update u^{k+1} according to (8).
- Interchange boundary values.

Non-linear elliptic equation

$$-\left(k(u)u'\right)' = f$$

Finite difference approximation

$$F(u)u = f$$

Newtons method; solve linear systems of the form

$$Au = f$$

in each Newton iteration.

Similar procedure for

$$-\nabla \cdot (k(u)\nabla u) = f.$$

Summary

All PDE's are solved using methods based on steps like

- Inner products
- vector + scalar$*$vector
- Matrix-vector multiplications
- Simple linear or non-linear updates

All of these operations are, in principle, easy to parallelize. We therefore conclude that PDE's are well-suited for parallel computers.