FARABI UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE



Lecture 7. Parallel Finite Differences

Dr. Bazargul Matkerim, Professor-Researcher

Outline

1. **The 1D heat equation**
   temperature in a thin metal rod
2. **Finite difference method**
   mesh, explicit scheme, serial code
3. **Parallelization with MPI**
   work division, ghost points, deadlock
4. **Jacobi method**
   steady-state case, MPI_Allreduce

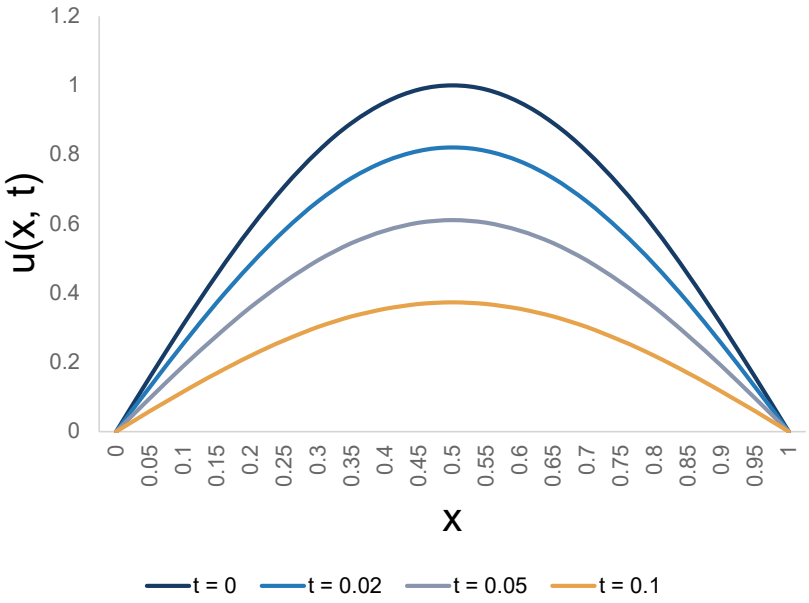


Figure: own computation, κ = 1, f = 0, I(x) = sin(πx)

The Heat Equation

Example: temperature of a **thin metal rod**

Initial temperature is known: **u(x, 0) = I(x)**

Both ends: **u(0, t) = u(1, t) = 0**

κ: heat conductivity, f(x, t): heat source



hot in the middle, cold at the ends

1D heat equation

$$\frac{\partial u}{\partial t} = \kappa \frac{\partial^2 u}{\partial x^2} + f(x,t)$$

The Finite Difference Method

Uniform mesh in space and discrete **time levels**

$$x_i = i\Delta x, \qquad \Delta x = \frac{1}{n}; \quad t_\ell = \ell \Delta t, \qquad \Delta t = \frac{T}{m}$$

Notation: **u** at point **x_i** and time level **t_ℓ**

Derivatives are approximated by **finite differences**

Finite differences

$$\frac{\partial u}{\partial t} \approx \frac{u_i^{\ell+1} - u_i^{\ell}}{\Delta t} \qquad \frac{\partial^2 u}{\partial x^2} \approx \frac{u_{i-1}^{\ell} - 2u_i^{\ell} + u_{i+1}^{\ell}}{\Delta x^2}$$

An Explicit Scheme for the 1D Heat Equation

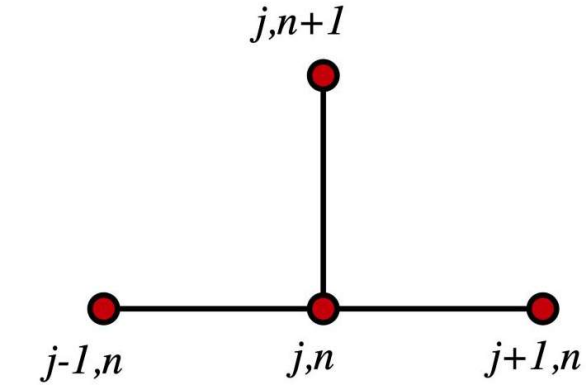
Explicit scheme

$$u_i^{\ell+1} = u_i^{\ell} + \kappa \frac{\Delta t}{\Delta x^2}(u_{i-1}^{\ell} - 2u_i^{\ell} + u_{i+1}^{\ell}) + \Delta t f(x_i, t_\ell)$$

For all **inner points** i = 1, 2, …, n − 1

Boundary: u_0 = u_n = 0

New value depends only on **three old values**



Source: Explicit method stencil, commons.wikimedia.org (public domain)

Serial Implementation

```
t = 0;
while (t < T) {
  x = dx;
  for (i = 1; i < n; i++) {
    u[i] = u_prev[i]
      + kappa*dt/(dx*dx)*(u_prev[i-1]-2*u_prev[i]+u_prev[i+1])
      + dt*f(x,t);
    x += dx;
  }
  u[0] = u[n] = 0.;                /* boundary condition */
  tmp = u_prev; u_prev = u; u = tmp; /* swap pointers */
  t += dt;
}
```

Source: Lecture 12 material “Parallel finite differences”, p. 8

Two arrays:

- u: new level $\ell+1$
- u_prev: old level ℓ

Main work: **time-stepping** loop

Swap of **pointers**, no copying

Parallelism and Work Division

New values are **independent** of each other

Divide the n − 1 inner points **evenly** among P processors

Maximum difference in work load is **1**

Blockwise decomposition: contiguous subsets

Points of processor p and start index

$$n_p = \lfloor \frac{n-1}{P} \rfloor + 1 \; if p < mod(n-1, P), \qquad else \lfloor \frac{n-1}{P} \rfloor$$

$$i_{start,p} = 1 + p\lfloor \frac{n-1}{P} \rfloor + min(p, mod(n-1, P))$$

Ghost Points and Communication

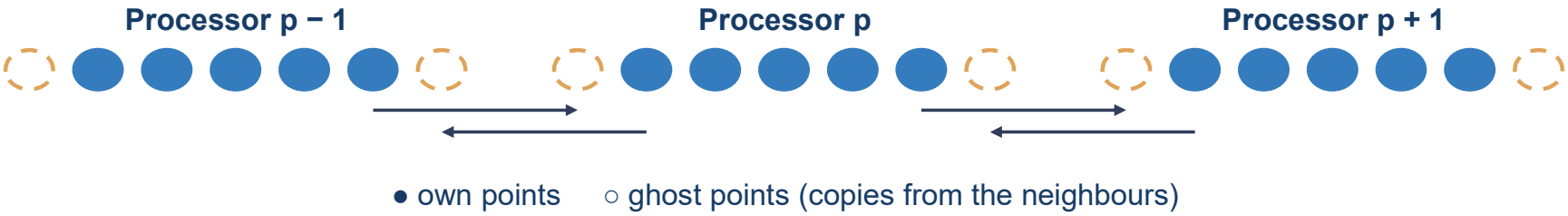
Local arrays of length **n_p + 2**

u_prev_local[0] from the left neighbour

u_prev_local[n_p+1] from the right neighbour

Two one-to-one exchanges per time step

Processor 0 has no left neighbour, P − 1 no right neighbour



MPI Communication and Deadlock

Both neighbours start with MPI_Recv: **deadlock**

Safest: **red-black** coloring
- odd rank (red): first Send, then Recv
- even rank (black): first Recv, then Send

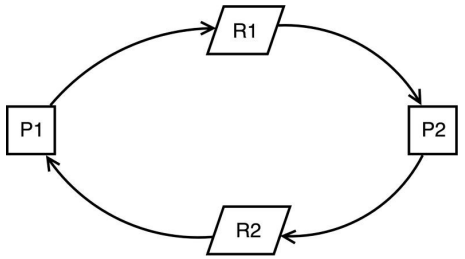
MPI_PROC_NULL if there is no neighbour

Non-blocking MPI_Isend/Irecv: **overlap** of communication and computation

One-to-one exchange in one call

```
MPI_Sendrecv(void *sendbuf, int sendcount,
  MPI_Datatype sendtype, int dest, int sendtag,
  void *recvbuf, int recvcount,
  MPI_Datatype recvtype, int source, int recvtag,
  MPI_Comm comm, MPI_Status *status);
```

Source: Lecture 12 material “Parallel finite differences”, p. 16



Source: Process deadlock, commons.wikimedia.org (FAL)

Stationary Heat Conduction and the Jacobi Method

Steady state: **u_t = 0**, f = f(x)

$$-\frac{d^2u}{dx^2} = g(x), \qquad g(x) = \frac{f(x)}{\kappa}$$

Jacobi method: u⁰, u¹, u², … from an initial guess

Similar to pseudo time stepping

Stop when the **difference** between iterations is small

Jacobi iteration

$$u_i^{\ell+1} = \frac{\Delta x^2 g(x_i) + u_{i-1}^{\ell} + u_{i+1}^{\ell}}{2}$$

Stopping criterion

$$diff = \sqrt{\sum_{i=1}^{n-1} (u_i^{\ell+1} - u_i^{\ell})^2}$$

Parallel Jacobi Method

Same parallelism and data partitioning as the **explicit scheme**

Two arrays on each processor: u_local, u_prev_local

Difference:

- each processor computes its own diff_p
- all diff_p are added with MPI_Allreduce
- then the square root is taken

Needs **one-to-one** and **collective** communication

```
double diff_p = 0, diff;
for (i = 1; i <= n_p; i++)
  diff_p += (u_local[i]-u_prev_local[i])
          * (u_local[i]-u_prev_local[i]);
MPI_Allreduce(&diff_p, &diff, 1,
              MPI_DOUBLE, MPI_SUM,
              MPI_COMM_WORLD);
diff = sqrt(diff);
```

Code sketch of the steps on p. 21

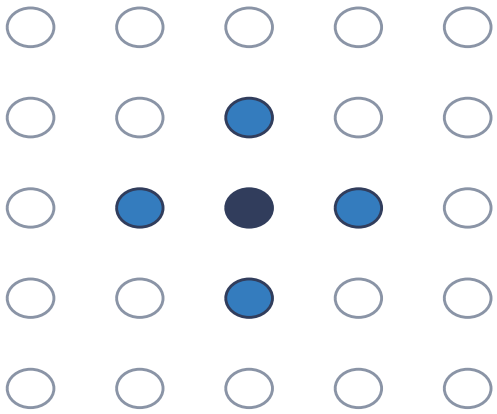
Comments and Exercise

More advanced solvers than Jacobi exist; parallelization is **not necessarily more difficult**

2D/3D heat equations: the same principles

Exercise: MPI implementation of the **Jacobi method** for a 2D steady-state heat equation

Reading: M. J. Quinn, **Parallel Programming in C with MPI and OpenMP**, Chapter 13



2D stencil: new value from 4 neighbours