Parallelisation of Grid-oriented Problems

Grid-oriented problems

- PDEs, image processing,… : data set defined on a *grid*
  local computations with small 'stencils'
  –> data dependencies between *neighbouring* grid points
- *grid point :* generic name for data associated with
  grid point, pixel, cell, finite element, …
- grid, data set & associated work: *partitioned in subdomains*
  the subdomains are assigned (*mapped*) to processors

Grid-oriented problems (cont.)

extra tasks (compared with sequential code)

- *partitioning & mapping* to ensure work load balance and communication minimisation
- *communication* between *neighbouring* subdomains

Model problems

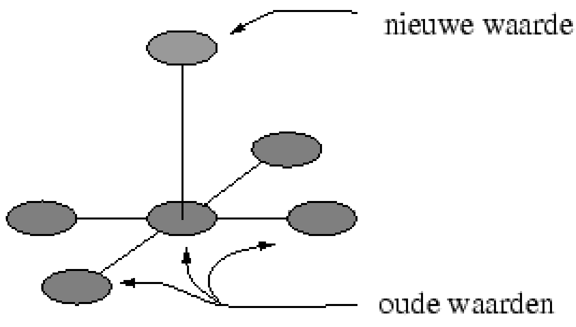
- PDEs
  - explicit time integration (forward Euler)
  - relaxation methods (Jacobi, Gauss-Seidel, SOR, …)

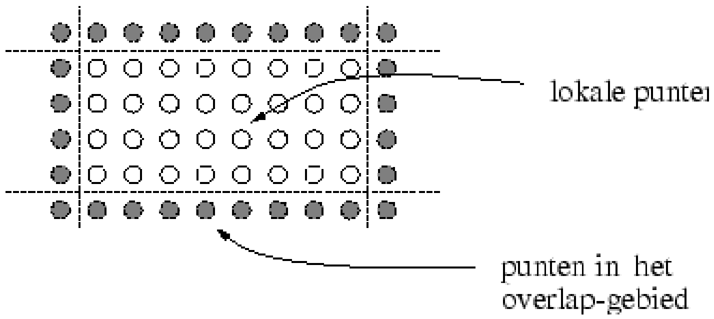
  on a structured (regular) 2D grid
- cellular automata (e.g. game of life)
- image processing
  - convolution on a 2D pixel matrix

same data-dependency pattern

-> same parallelisation strategy

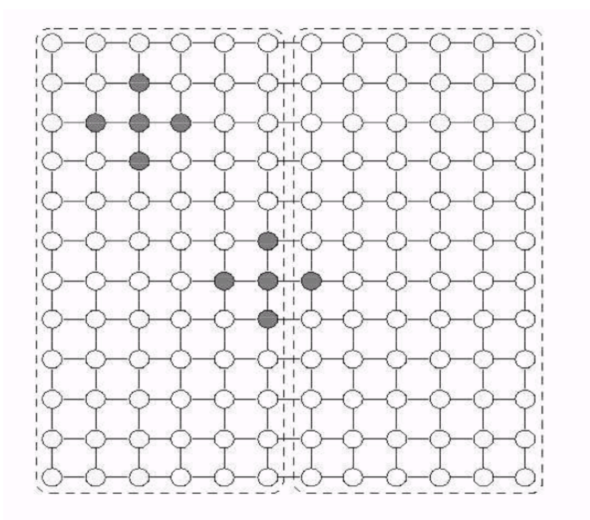
Explicit time integration & convolution



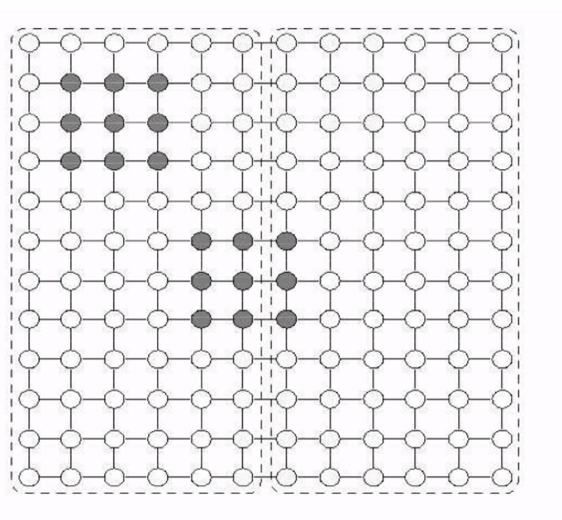


Computational ‘molecules’

5 point stencil

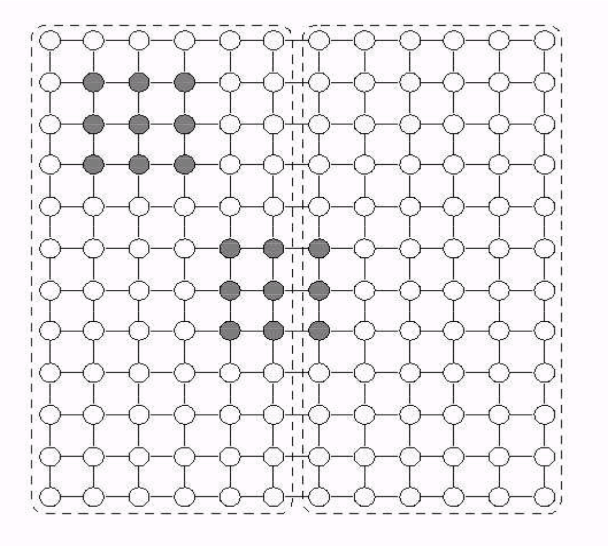


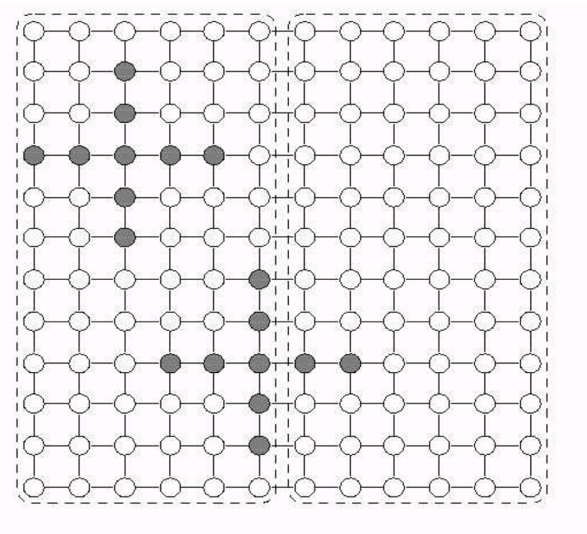
9 point stencil



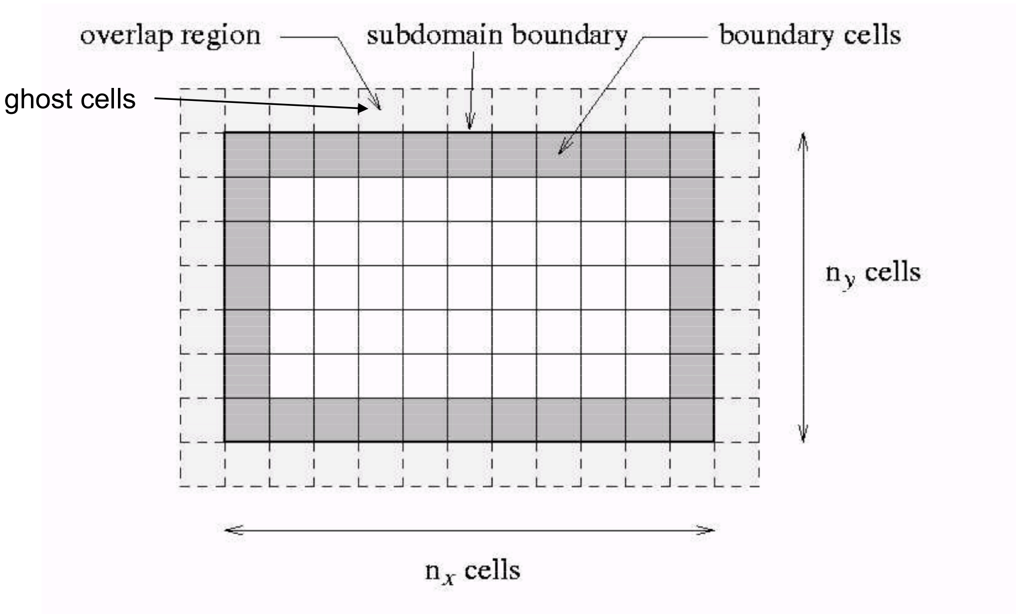
Computational 'molecules' (cont.)

- two different 9 point stencils





Subdomains & overlap regions



Note: overlap region can have a width > 1

Skeleton of a typical program

in every subdomain (processor):

- exchange data in the overlap region

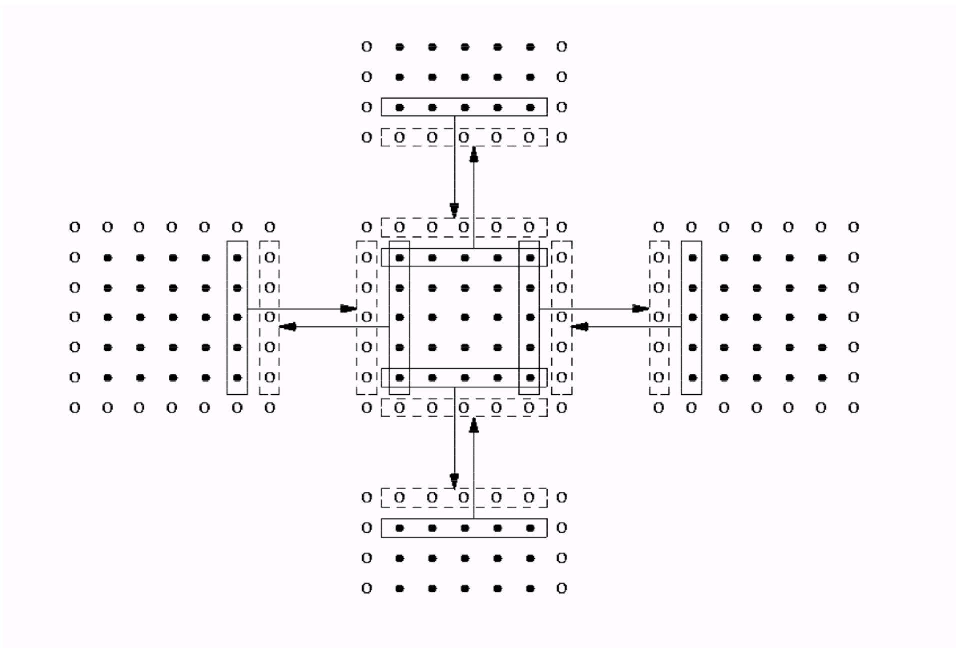
  communication with procs. holding neighbouring subdomains

- do calculations for all grid points in subdomain
- check for stopping criterion (e.g. convergence check)

  global communication (reduction or all-reduce)

Exchange overlap regions

5 point stencil



Analysis of communication overhead

- assume p processors; $n = n_x \times n_y$ points per subdomain;
- *only* communication overhead; no sequential part; no load imbalance

$$T(p) = T_{calc} + T_{comm}; \quad T(1) = pT_{calc}$$

$T(p)$: parallel execution time ; $T(1)$: execution time on 1 proc.

T_{calc} : calculation time on *each* proc ; T_{comm} : communication time

Speedup $$S(p) = \frac{pT_{calc}}{T_{calc} + T_{comm}} = \frac{p}{1 + \frac{T_{comm}}{T_{calc}}} = \frac{p}{1 + f_c}$$

Efficiency $$E(p) = \frac{1}{1 + f_c} \approx 1 - f_c$$

Analysis of communication overhead (cont.)

- Communication overhead *relative to calculation cost !* $$f_c = \frac{T_{comm}}{T_{calc}}$$
- For the model problem $T_{calc} = c_f\, n_x n_y t_{calc} \quad (t_{calc} = 1/r)$

$$T_{comm} = c_c 2(n_x + n_y)\, t_{comm} \quad (t_{comm} = g)$$

t_{calc} : time to perform a floating point operation
t_{comm}: average time to communicate one floating point number

Analysis of communication overhead (cont.)

Communication overhead $$f_c = \frac{T_{comm}}{T_{calc}} = \frac{c_c}{c_f} \frac{2(n_x + n_y)}{n_x n_y} \frac{t_{comm}}{t_{calc}}$$

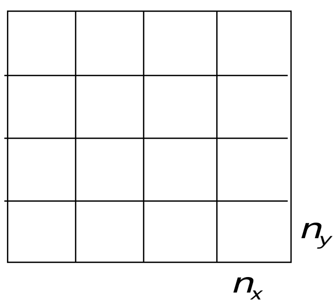
depends on :

- the *size of the subdomain*: large subdomains have a small *perimeter to surface ratio*
- the *machine characteristic t_{comm}/t_{calc}* : indicates how fast communication can be performed compared with floating point operations
- the *algorithm* via the *ratio c_c/c_f* : f_c is small when many flops per grid point (c_f) compared with the amount of data associated with a grid point (c_c)

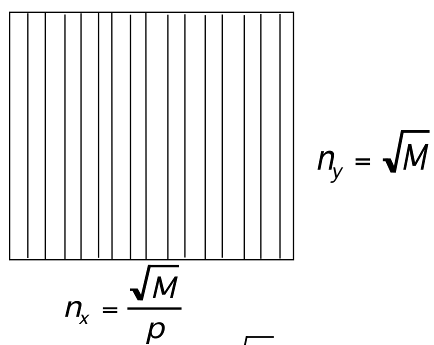
Partitioning strategies

2D $\sqrt{M} \, x \sqrt{M}$ grid: M grid points; $n = M/p$ grid points per proc.

2D (blockwise) partitioning



1D (stripwise) partitioning



$n = n_x \, x \, n_y$; square blockwise partitioning if $n_x = n_y = \sqrt{n}$

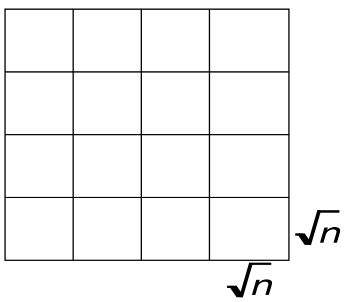
Partitioning strategies (cont.)

- communication volume ~ perimeter of subdomain
  square subdomains ($n_x = n_y$): minimal perimeter (for same area)
  2D ***square*** *(blockwise) partitioning* is to be preferred
- BUT:
  1D (stripwise) partitioning:
  – higher communication volume
  – fewer neighbours –> fewer messages
  choice : depends on problem & machine characteristics
- *1D partitioning* may be better also when communication mainly in one direction (*anisotropic* communication)

Comm. overhead: dependence on problem size

- 2D $\sqrt{M}\, x \sqrt{M}$ grid: M grid points; $n = M/p$ grid points per proc.

blockwise partitioning



per proc: $\sqrt{n}\, x \sqrt{n}$ points

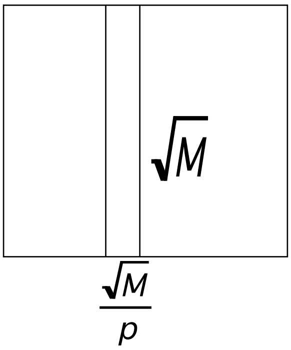
$$f_c = \frac{T_{comm}}{T_{calc}} \propto \frac{\sqrt{n}}{n} = \frac{1}{\sqrt{n}} = \frac{\sqrt{p}}{\sqrt{M}}$$

- f_c (and speedup & efficiency) is constant when n (problem size per proc) is constant (i.e. M grows linearly with p)
- $f_c \uparrow$ (speedup & efficiency$\downarrow$) when M is constant and p grows

Comm. overhead: dependence on problem size

- 2D $\sqrt{M}\ x \sqrt{M}$ grid: M grid points; $n = M/p$ grid points per proc.

  stripwise partitioning



per proc: $\sqrt{M}\ x \frac{\sqrt{M}}{p}$ points

$$f_c = \frac{T_{comm}}{T_{calc}} \propto \frac{\sqrt{M}}{n} = \frac{\sqrt{p}}{\sqrt{n}} = \frac{p}{\sqrt{M}}$$

- f_c ↑(speedup & efficiency ↓) when n is constant and p grows
- f_c ↑↑ (speedup & efficiency ↓↓) when M is constant and p grows

Comm. overhead: dependence on problem size

- ***3D problems***

  communication overhead

  $\propto$ 'surface to volume' ratio of the subdomains

blockwise partitioning: $n = n^{1/3}\ x\ n^{1/3}\ x\ n^{1/3}$ points per proc.

- f_c decreases slower as function of increasing n than in 2D case BUT typically n is much larger in 3D than in 2D
- For squares (2D) and cubes (3D): $f_c \propto$ *1/(number of points per direction)* ($1/n^{1/2}$, *resp.* $1/n^{1/3}$)
- *d-dimensional problems :* $f_c \propto 1/n^{1/d}$