A Short Course in Parallel Scientific Computing

Technical University of Budapest

Graham Horton, University of Erlangen-Nürnberg

Talk 3 : Solution of elliptic PDEs

Inhalt

- Partial differential equations - examples
- Discretisation
- Iterative solution of the algebraic equations
- The Jacobi method
- The Gauß-Seidel method
- Three efficiency traps

Partial Differential Equations

Partial differential equations describe many phenomena

- Convective and diffusive transport
- Fluid flow
- Chemical reactions
- Electrical and magnetic fields

For this reason they are very important in Science and Engineering

Most compute-intensive problems are differential equations

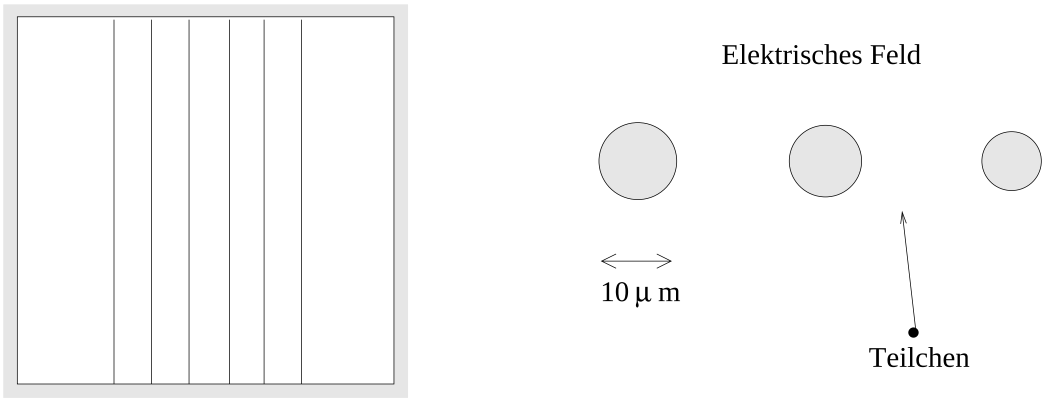
Partial Differential Equations

Grand Challenges:

Year	Field	GFLOPS
1980	2-D plasma modeling	0.1
	Airfoil design	0.6
	48-hour weather	1
	chemical dynamics	1
1990	3-D plasma modeling	0.6
	estimates of Higgs boson mass	1
	72-hour weather	6
	ultralarge scale IC design	10
	vehicle signature	20
2000	speech and natural language	8
	pharmaceutical design	10
	structural biology	20
	climate modeling	1000
	fluid turbulence	1000
	pullution dispersion	1000
	human genome	1000
	ocean circulation	1000
	quantum chromodynamics	1000
	semiconductor modeling	1000
	superconductor modeling	1000
	combustion systems	1000
	vision and cognition	1000

Example - Particle Physics

In an accelerator the following apparatus is used to detect particles



A row of gold wires generates an electric field

A particle entering the field changes it and can thus be detected

Problem: Layout of the wires for maximally homogeneous field

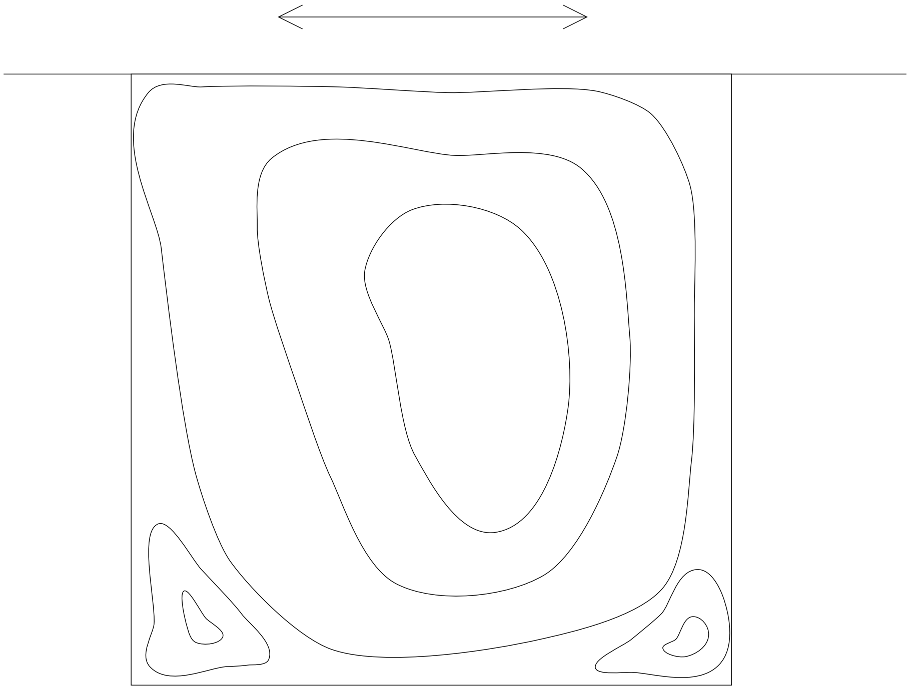
Solution: Numerical solution of the p.d.e.

$$-\frac{\partial^2 \phi}{\partial x^2} - \frac{\partial^2 \phi}{\partial y^2} = 0$$

with given values of ϕ on the boundaries

Example - Fluid Mechanics

Flow in a closed square container:



Oscillating lid induces fluid flow

Numerical solution of the Navier-Stokes equations

$$\begin{aligned}
\frac{\partial(\rho u)}{\partial t} + \frac{\partial}{\partial x}(\rho u^2 - \mu\frac{\partial u}{\partial x}) + \frac{\partial}{\partial y}(\rho u v - \mu\frac{\partial u}{\partial y}) &= -\frac{\partial p}{\partial x} + f_x \\
\frac{\partial(\rho v)}{\partial t} + \frac{\partial}{\partial x}(\rho u v - \mu\frac{\partial v}{\partial x}) + \frac{\partial}{\partial y}(\rho v^2 - \mu\frac{\partial v}{\partial y}) &= -\frac{\partial p}{\partial y} + f_y \\
\frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} &= 0
\end{aligned}$$

Numerical Solution

Partial differential equations are almost never analytically solvable

Therefore we attempt an approximate numerical solution

Discretize the domain of the PDE

Compute an approximate solution in the grid points

Yields a finite number of discrete solution values

Replace partial derivatives by differences

i.e. transform PDE into a system of algebraic equations

Numerical Solution

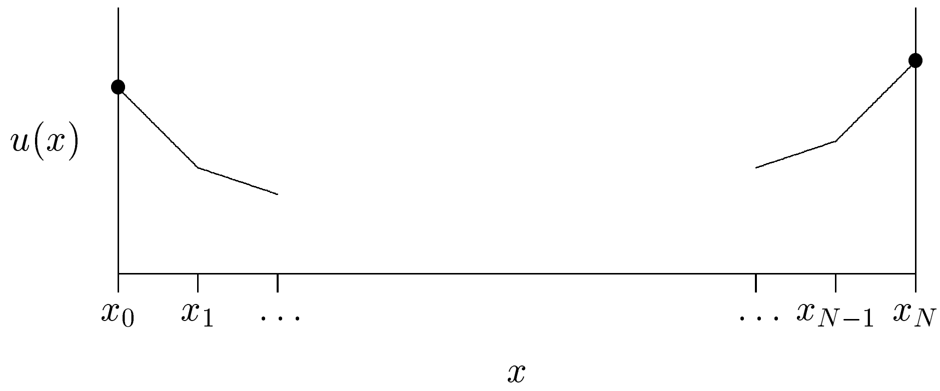
Consider as an example the 1-D linear PDE

$$-\frac{\partial^2 u}{\partial x^2} = f(x), \quad x = (0, 1)$$

with the continuous solution $u(x)$

The boundary values of $u(x)$ at $x = 0$ and $x = 1$ are known

Discretize the interval $x = (0, 1)$ using $N + 1$ equidistant points x_i



Mesh size $x_{i+1} - x_i = h$

In general, different mesh sizes can be used

Look at u and f only in the points x_i ($= u_i$, f_i)

Discretisation

Consider the Taylor series approximation to $u(x)$ around $x+h$ and $x-h$:

$$\begin{aligned} u(x+h) &= u(x) + hu'(x) + \frac{h^2}{2}u''(x) + O(h^3) \\ u(x-h) &= u(x) - hu'(x) + \frac{h^2}{2}u''(x) + O(h^3) \end{aligned}$$

Rearrange:

$$h^2 u''(x) = u(x+h) + u(x-h) - 2u(x) + O(h^3)$$

This yields

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_i \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}$$

We have turned the partial derivative into a linear combination of neighbouring gridpoint values

This gives a system of $N-1$ equations for the inner gridpoints

$$\frac{1}{h^2}\begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & & & & \\ & & & & -1 & 2 \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{bmatrix} = \begin{bmatrix} f_1 + \frac{1}{h^2}u_0 \\ f_2 \\ \vdots \\ f_{N-2} \\ f_{N-1} + \frac{1}{h^2}u_N \end{bmatrix}$$

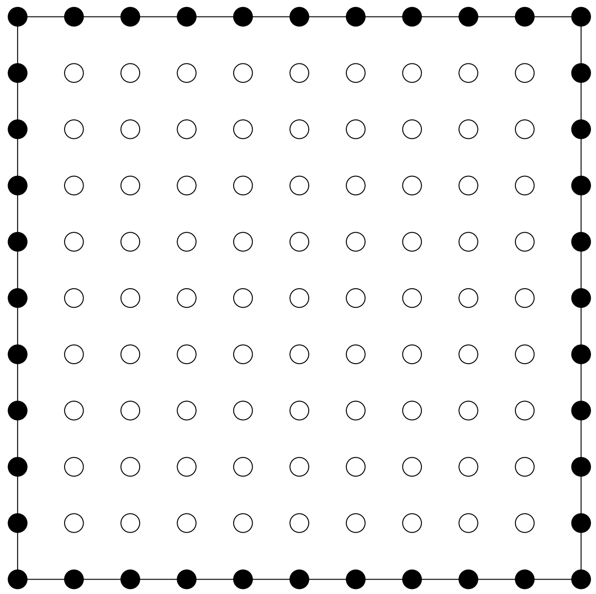
The boundary values are known and go into the right hand side

Discretisation

Similarly the 2-D heat equation

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = f(x,y), \quad x \subset (0,1), y \subset (0,1)$$

can be discretized by the equidistant square grid

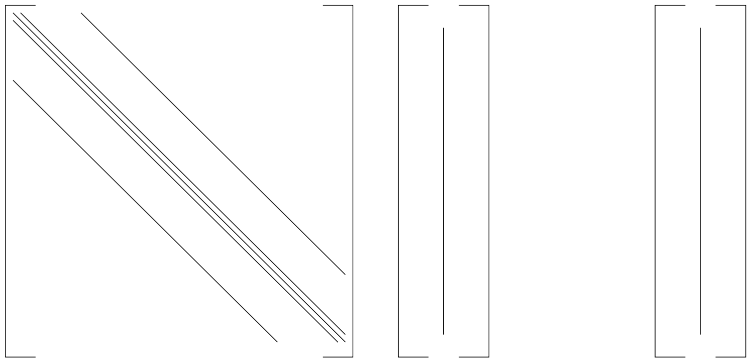


We obtain as discretisation in every inner point (i,j)

$$-\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)\bigg|_{i,j} \approx \frac{-u_{i-1,j} + 2u_{i,j} - u_{i+1,j}}{h^2} + \frac{-u_{i,j-1} + 2u_{i,j} - u_{i,j+1}}{h^2}$$

Discretisation

Using *lexicographic* ordering, the system has the following structure



A u = f

The main diagonal has the value $4/h^2$, otherwise $-1/h^2$

The matrix is of size $(N-1)(N-1)$

It has only five non-zero diagonals

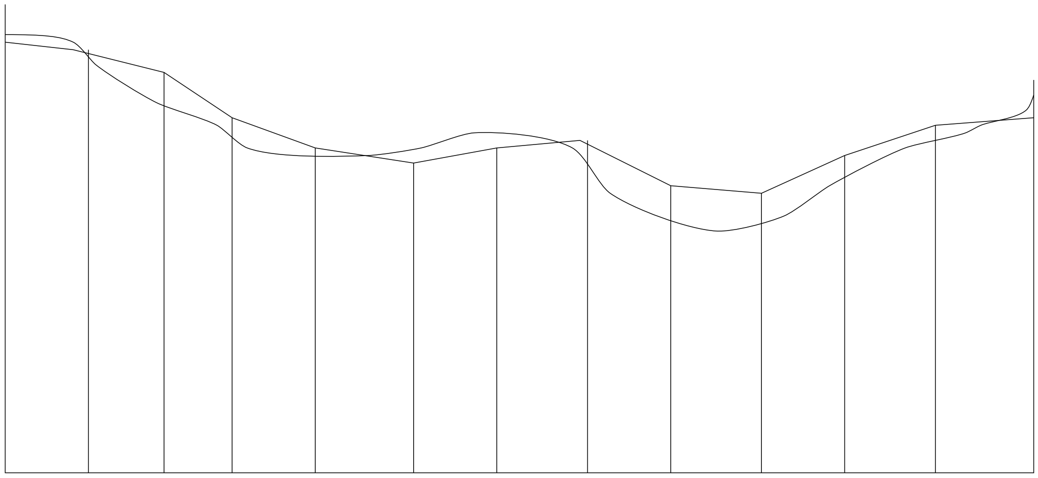
We have therefore replaced our PDE by a large linear system of equations

Questions: What problems does this bring us ?

- How do we solve the systems?
- How exact is the solution (discretisation error!)?
- The solution is only defined in the grid points

Discretisation Error

After we have solved the system $Au = f$ we have the following situation:



The discrete solution differs from the continuous analytic solution

The discretisation error depends on h, so ...

We can reduce it by increasing the number of grid points

High accuracy requires a very large number of grid points

High accuracy $\Rightarrow$ high compute and memory requirements!

Iterative Methods

We usually prefer iterative methods for the solution of $Au = f$

Why: Need less memory, and are faster and more stable

Find matrices M and N with $A = M + N$ and M easily invertible

Choose starting vector $u^{(0)}$

Perform the iteration

$$Mu^{(n)} = f - Nu^{(n-1)}$$

If the iteration converges, we have

$$u^{(\infty)} = u$$

and

$$Mu = f - Nu \qquad \text{oder} \quad Au = f$$

The error in the current iterate is given by the defect (residuum)

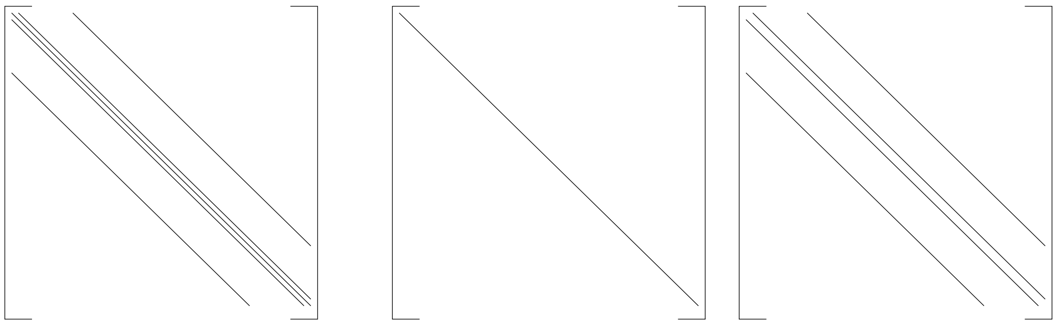
$$|d^{(n)}| = |f - Au^{(n)}|$$

i.e. the difference between left and right sides of $Au^{(n)} = f$

Iterative Methods

The simplest iterative method is Jacobi's method

Choose M as main diagonal of A and N as the rest



A **M** **N**

The iteration is easy to perform

$$Mu^{(n+1)} = f + Nu^{(n)}$$

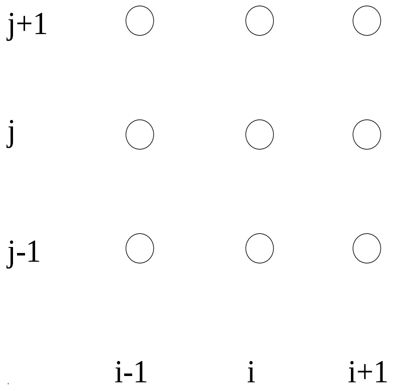
beacuse the inversion of a diagonal matrix is trivial

```
while error > ε do
   for i = 1 to (N − 1) ∗ (N − 1) do
       u[i]^new = 0.25*(h*h*f[i]+u[i-1]+
                  u[i+1]+u[i-n]+u[i+n])
   u = u^new
```

In practice we never use this notation

Iterative Methods

The data is always stored in the grid



Sequential program for the Jacobi iteration:

```
while error > ϵ do
   for i = 1 to N−1 do
       for j = 1 to N−1 do
           u[i,j]^new = 0.25*(h*h*f[i,j]+u[i-1,j]+
                        u[i+1,j]+u[i,j-1]+u[i,j+1])
   u = u^new
```

This runs very efficiently on all parallel computers

It needs no modification

Parallelisation on MIMD Computers

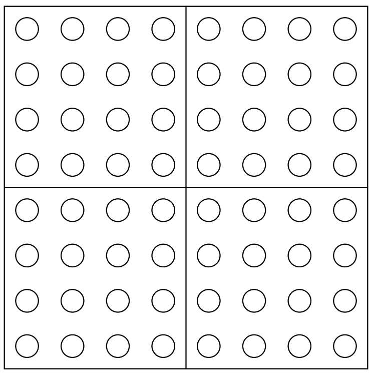
All operations of an iteration are independent

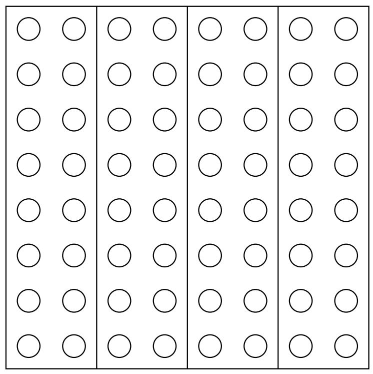
Distribution of grid points is in principle arbitrary

(Assuming equal load!)

Goal : Minimisation of communication

Solution: Checkerboard or strip partitioning





This is one of the Meiko programming examples!

Optimal Grid Partitioning

Denote grid sidelengths by $N_x = N_y$

Number of processors is p

Latency for a message is α

Transfer time for one FP value is β

Then the checkerboard partitioning has

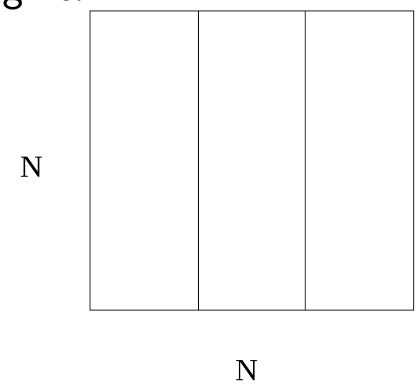
$$T_{Comm} = 4(\alpha + \frac{N_x}{\sqrt{p}}\beta)$$

and the strip partitioning has

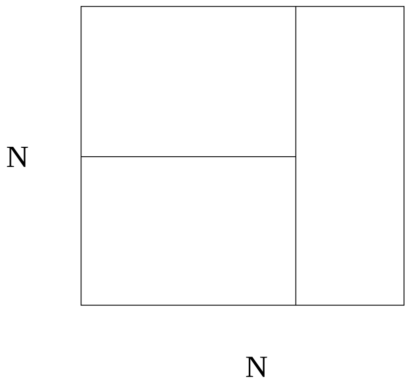
$$T_{Comm} = 2(\alpha + N_x\beta)$$

Therefore the following are optimal for a square grid:

For large α :



For small α :



”IΛ empfehle Ihnen diesen Modu‘ zur NaΛ-ahmung. SΛwerliΛ werϒn Sie je wieϒr direΠ eliminiren, wenig=en‘ niΛt, wenn Sie mehr al‘ zwei UnΔkannte ΨΔn. Da‘ indireΠe Verfahren læßt siΛ Ψlb im SΛlafe au‘fuehren oϒr man kann wæhrend ϒ*elΔn an anϒre Ding ϒnken”.

C.F. Gauß

The Gauß-Seidel Method

We obtain the Gauß-Seidel method by the choice

$$M = D + L, \qquad N = U$$

i.e.

$$(D + L)u^{(n+1)} = f - Uu^{(n)}$$

Here newly computed results are used immediately

$D + L$ is triangular $\Rightarrow$ easily solvable system

BUT: The solution of a triangular matrix contains a recursion

The Gauß-Seidel Method

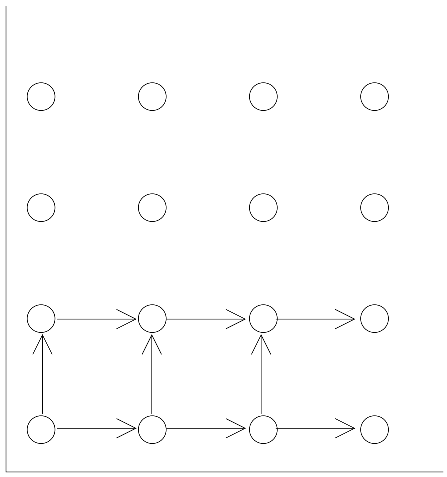
For the model problem we get the following algorithm

```
procedure gauss-seidel
  for i=1 to n-1 do
    for j=1 to n-1 do
      u[i][j] = 0.25*h*h*(u[i-1][j]+u[i+1][j]+
                          u[i][j-1]+u[i][j+1])
end
```

We no longer need the temporary variable

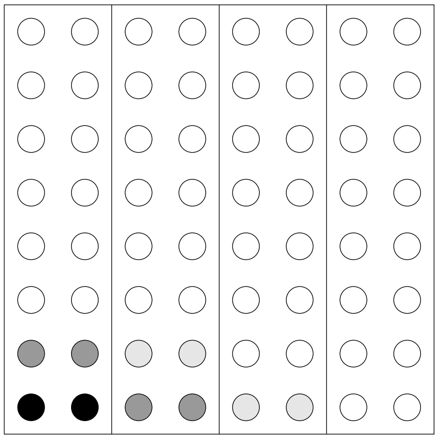
BUT: We have a data dependency

$u_{i,j}$ must be computed after $u_{i-1,j}$ and $u_{i,j-1}$



Parallelisierung des Gauß-Seidel-Verfahrens

Only the strip partitioning is possible



We have $1 \leq p \leq n$

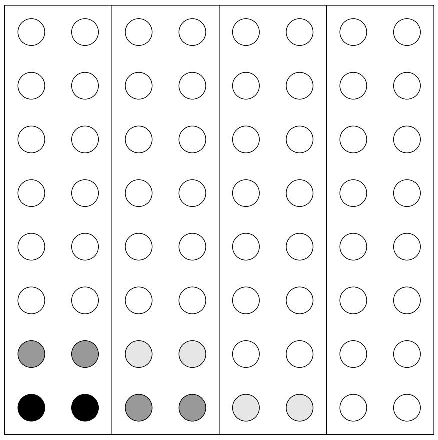
There is a startup phase, until all processors are working

Then a parallel phase

Not optimal parallelism, but

Converges faster than the Jacobi method

Speedup of the Gauß-Seidel Method



Let computation time per grid point be T_R

Let communication time per grid point be T_K

Sequential time: $T_S = n^2 T_R$

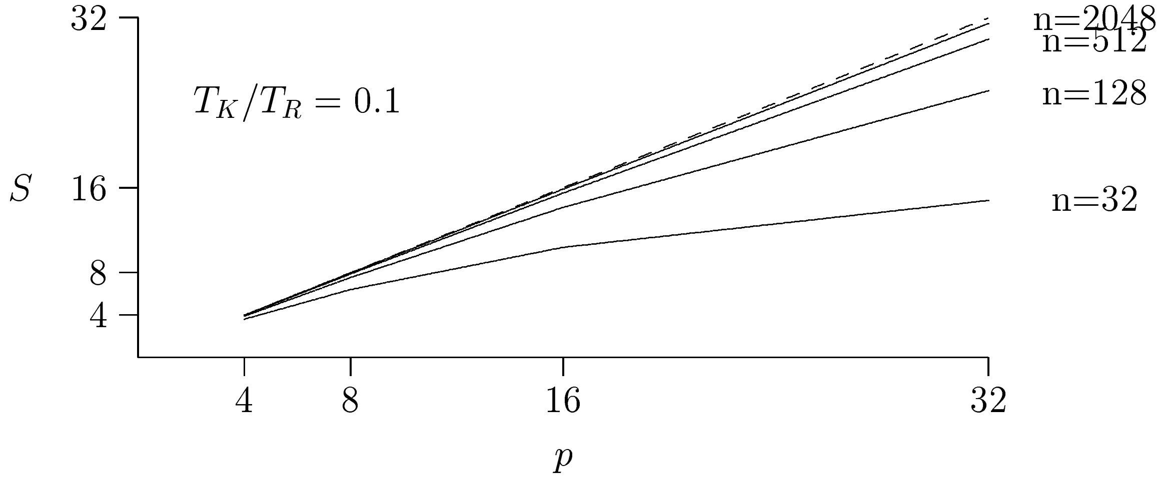
Startup time : $T_E = (p-1)T_K + (n - \frac{n}{p})T_R$

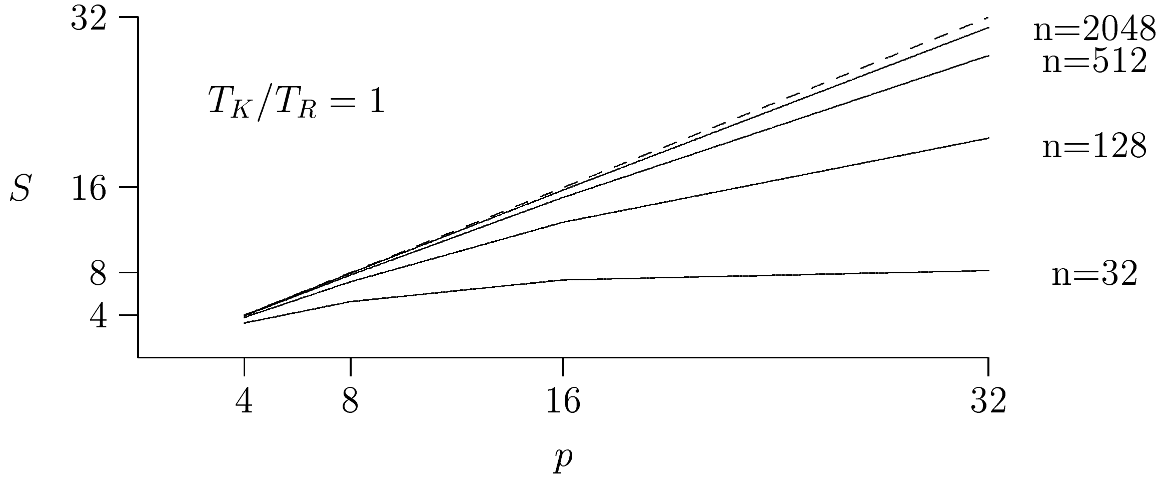
Parallel phase time : $T_P = (n-1)T_K + n\frac{n}{p}T_R$

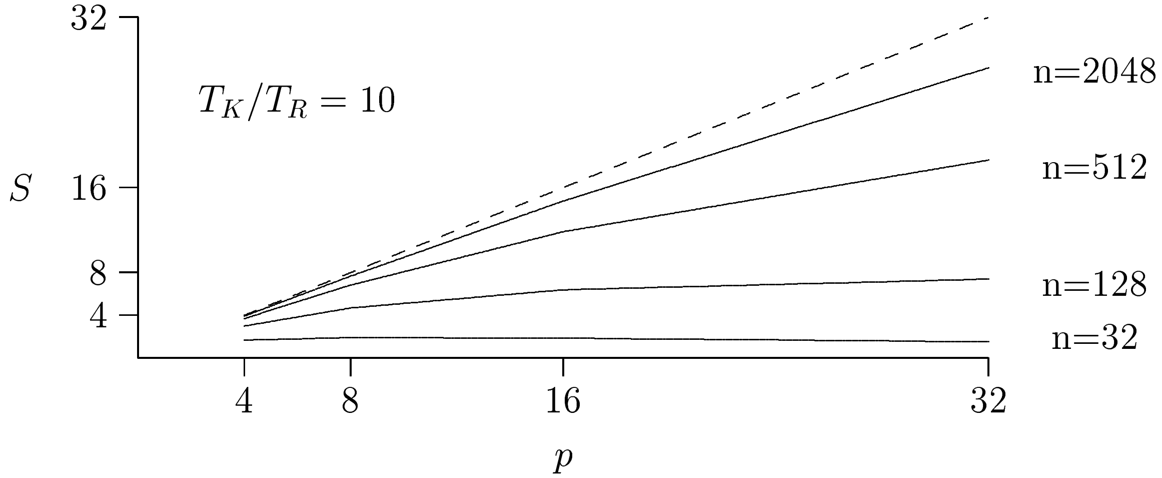
Speedup :

$$S = \frac{T(1)}{T(p)} = \frac{n^2 T_R}{(p-1)T_K + (n - \frac{n}{p})T_R + (n-1)T_K + n\frac{n}{p}T_R}$$

Speedup of the Gauß-Seidel Method



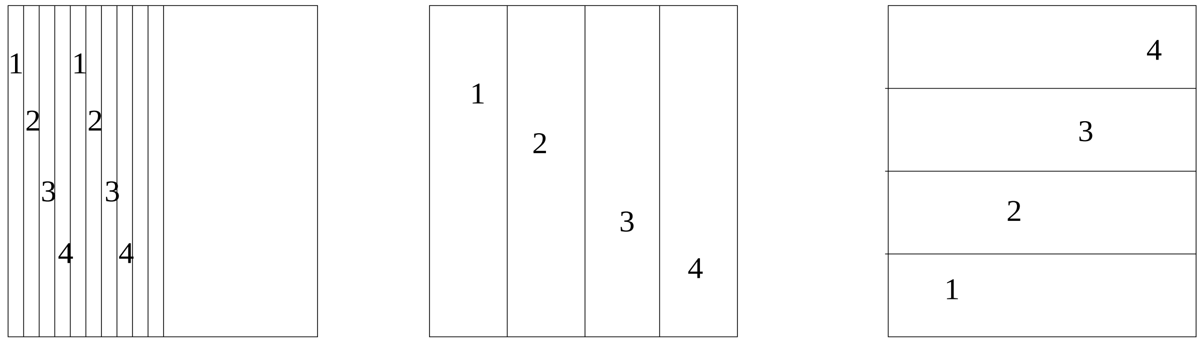




Three Efficiency Traps

The results of the parallel programming exercise vary substantially

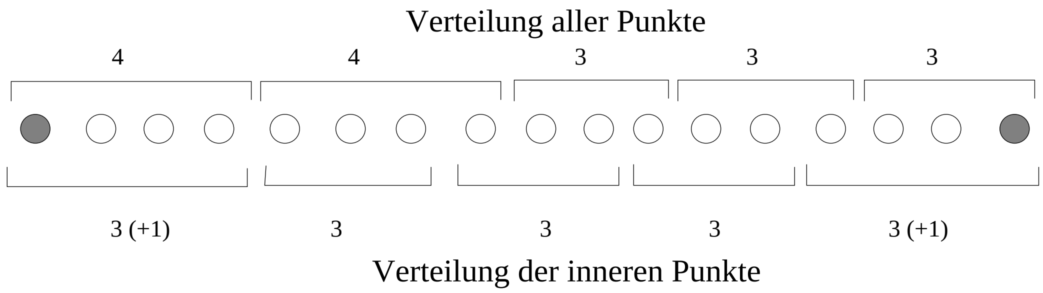
Consider three different partitionings of the grid:



(Unexpected) low efficiency of the Jacobi method due to

- Bad partitioning : too high communication cost
- Non-optimal communications programming : must be parallel !
- Bad load balancing : Boundary points are not computed !
  $\Rightarrow$ Equal partitioning of the *inner* grid points

Compare partitionings of 17 grid points onto 5 processors:


Verteilung aller Punkte
4
4
3
3
3
3 (+1)
3
3
3
3 (+1)
Verteilung der inneren Punkte

Three Efficiency Traps

Communications programming :

```
cwrite(toright, .)
cread(fromleft, . )
cwrite(toleft, .)
cread(fromright, .)
```

This runs *totally sequentially!*

$\Rightarrow$ Use parallel strategy that differentiates by `procno % 2`

With the cyclic partitioning every grid point must be transferred!

$\Rightarrow$ leads to $\mathcal{O}(n^2)$ communication volume

Horizontal strips lead to many (n) messages :

```
for i = 1 to n
cwrite(up, & u[i][jmax], sizeof(double))
```

Better: use vertical strips (only one message)

```
cwrite(right, & u[imax][1], n*sizeof(double))
```