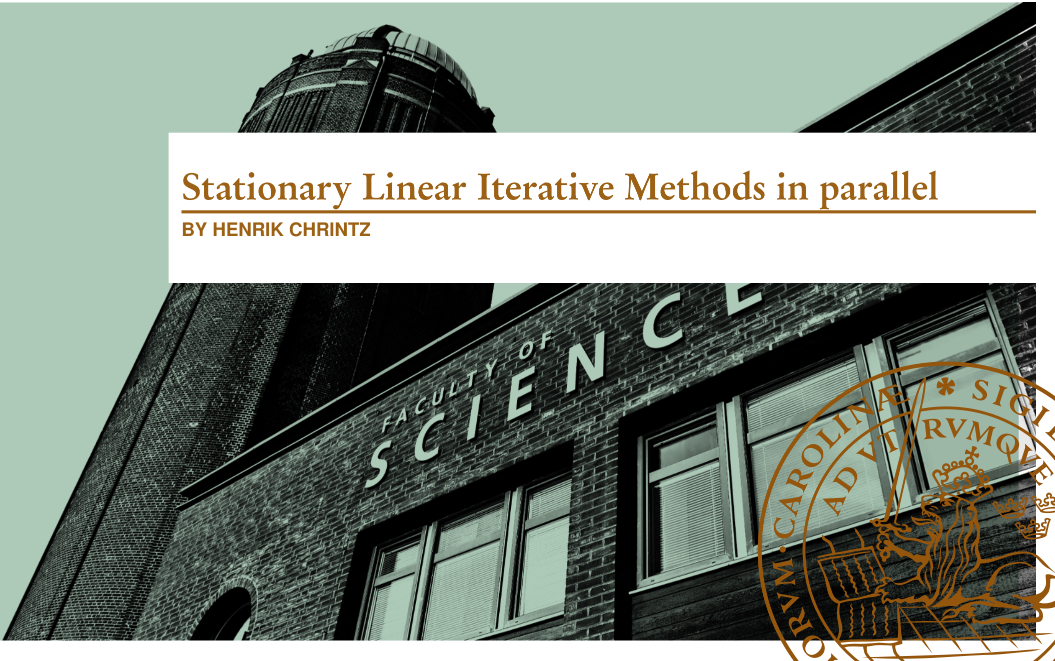
Stationary Linear Iterative Methods in parallel

BY HENRIK CHRINTZ



Disclaimers and corrections

- The picture in the title page is not mine. It was included in the LaTeX Beamer template and I assume it is OK to use it.
- All source code and the report can be found at: https://gitlab.com/Drunte/public-sor
- Sources can be found in the report.
- Most proofs will be omitted in this presentation, see the report instead.
- The equation at the end of section 5.4 in the report should be:

$$A_{RB} = PAP^T$$

- Theorem 4 should conclude: $\kappa(A) = \mathcal{O}(n^2)$

Introduction

- Partial differential equations

Introduction

- Partial differential equations
- PDE's are hard to solve exactly.

Introduction

- Partial differential equations
- PDE’s are hard to solve exactly.
- So solutions must be approximated instead.

Introduction

- Partial differential equations
- PDE's are hard to solve exactly.
- So solutions must be approximated instead.
- Poisson type is the target of investigation.

Introduction

- Partial differential equations
- PDE's are hard to solve exactly.
- So solutions must be approximated instead.
- Poisson type is the target of investigation.
- Approximated by a linear equation system.

Introduction

- Partial differential equations
- PDE’s are hard to solve exactly.
- So solutions must be approximated instead.
- Poisson type is the target of investigation.
- Approximated by a linear equation system.
- Options for solving said systems.

Introduction

- Partial differential equations
- PDE's are hard to solve exactly.
- So solutions must be approximated instead.
- Poisson type is the target of investigation.
- Approximated by a linear equation system.
- Options for solving said systems.
- Increased performance by working in parallel.

Introduction

- Partial differential equations
- PDE's are hard to solve exactly.
- So solutions must be approximated instead.
- Poisson type is the target of investigation.
- Approximated by a linear equation system.
- Options for solving said systems.
- Increased performance by working in parallel.
- Due to strong coupling: a remedy is Red-Black ordering.

Introduction

- Partial differential equations
- PDE's are hard to solve exactly.
- So solutions must be approximated instead.
- Poisson type is the target of investigation.
- Approximated by a linear equation system.
- Options for solving said systems.
- Increased performance by working in parallel.
- Due to strong coupling: a remedy is Red-Black ordering.
- Implementation details, experiment results and conclusions.

Poisson's equation

Poisson's equation is given by:

$$-\Delta u(x) = f(x), \qquad x \in \Omega \subset \mathbb{R}^n,$$

$$u(x) = g(x), \qquad x \in \partial\Omega.$$

Where u(x) is the solution, x is our independent variable, Ω is a compact subset of $\mathbb{R}^n$, $\partial\Omega$ is its boundary, f(x) describes the problem, g(x) is the boundary conditions and $-\Delta$ is the Laplace operator.

Discretization with a grid

- We want a linear system on the form: $A\mathbf{u} = \mathbf{f}$

Discretization with a grid

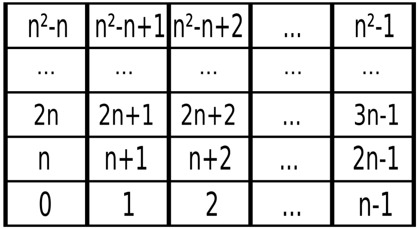
- We want a linear system on the form: $A\mathbf{u} = \mathbf{f}$
- First we partition Ω into an equidistant grid of cells.

Discretization with a grid

- We want a linear system on the form: $A\mathbf{u} = \mathbf{f}$
- First we partition Ω into an equidistant grid of cells.
-

n²-n	n²-n+1	n²-n+2	...	n²-1
...	...	...	...	...
2n	2n+1	2n+2	...	3n-1
n	n+1	n+2	...	2n-1
0	1	2	...	n-1

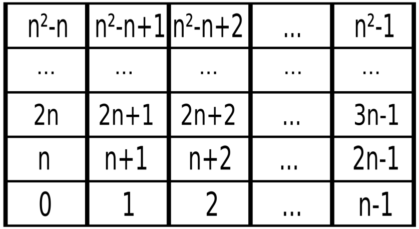
Discretization with a grid

- We want a linear system on the form: $A\mathbf{u} = \mathbf{f}$
- First we partition Ω into an equidistant grid of cells.
- 

n^2-n	n^2-n+1	n^2-n+2	...	n^2-1
...	...	...	...	...
2n	2n+1	2n+2	...	3n-1
n	n+1	n+2	...	2n-1
0	1	2	...	n-1

- Now **f** can be generated by evaluating f(x) over each cell center.

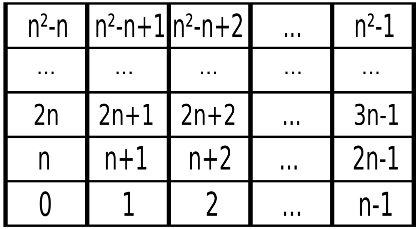
Discretization with a grid

- We want a linear system on the form: $A\mathbf{u} = \mathbf{f}$
- First we partition Ω into an equidistant grid of cells.
- 

n^2-n	n^2-n+1	n^2-n+2	...	n^2-1
...	...	...	...	...
2n	2n+1	2n+2	...	3n-1
n	n+1	n+2	...	2n-1
0	1	2	...	n-1

- Now **f** can be generated by evaluating f(x) over each cell center.
- Likewise, **u** will approximate u(x) at these cell centers.

Discretization with a grid

- We want a linear system on the form: $A\mathbf{u} = \mathbf{f}$
- First we partition Ω into an equidistant grid of cells.
- 

n^2-n	n^2-n+1	n^2-n+2	...	n^2-1
...	...	...	...	...
2n	2n+1	2n+2	...	3n-1
n	n+1	n+2	...	2n-1
0	1	2	...	n-1

- Now **f** can be generated by evaluating f(x) over each cell center.
- Likewise, **u** will approximate u(x) at these cell centers.
- The derivation of A comes next.

Symmetric difference approximation

For simplicity i chose to use a square equidistant grid and square $\Omega \subset \mathbb{R}^2$.
The $-\Delta$ operator can be approximated by a matrix in different ways. The one i chose was the symmetric central finite difference scheme. From the definition of the derivative we get:

- The two discrete derivatives: $f'(x) \approx \frac{f(x+h)-f(x)}{h}$ and $f'(x) \approx \frac{f(x)-f(x-h)}{h}$.

Symmetric difference approximation

For simplicity i chose to use a square equidistant grid and square $\Omega \subset \mathbb{R}^2$.
The $-\Delta$ operator can be approximated by a matrix in different ways. The one i chose was the symmetric central finite difference scheme. From the definition of the derivative we get:

- The two discrete derivatives: $f'(x) \approx \frac{f(x+h)-f(x)}{h}$ and $f'(x) \approx \frac{f(x)-f(x-h)}{h}$.
- Combining these two to approximate the second order derivative yields:

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

Symmetric difference approximation

For simplicity i chose to use a square equidistant grid and square $\Omega \subset \mathbb{R}^2$.
The $-\Delta$ operator can be approximated by a matrix in different ways. The one i chose was the symmetric central finite difference scheme. From the definition of the derivative we get:

- The two discrete derivatives: $f'(x) \approx \frac{f(x+h)-f(x)}{h}$ and $f'(x) \approx \frac{f(x)-f(x-h)}{h}$.
- Combining these two to approximate the second order derivative yields:

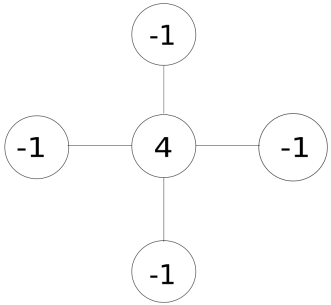
$$f''(x) \approx \frac{f(x+h)-2f(x)+f(x-h)}{h^2}$$

- Now taking second derivatives in two different directions and negating we get:

$$-\Delta u \approx \frac{-u(v_1, v_2+h)-u(v_1-h, v_2)+4u(v_1, v_2)-u(v_1+h, v_2)-u(v_1, v_2-h)}{h^2}$$

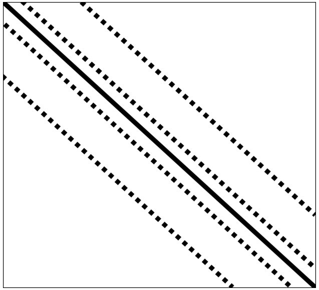
5-point stencil

This expression can be represented by a stencil after multiplying the h^2 factor.



Sparsity pattern

The resulting matrix has the following sparsity pattern:



The full line on the main diagonal has all 4's and the dotted lines are all -1.

For a grid of size $n \times n$ the matrix is of size $m \times m$ with $m = n^2$.

Preliminaries

Some notations.

- x_i : the i'th element of a vector x.
- $A_{i,j}$: the element of a matrix A, of row i and column j.
- A^{-1} : the inverse of a matrix A.
- A^T : the transpose of a matrix A.
- $u^T v$: the inner product of two vectors u and v.
- $x^{(k)}$: the k'th iterate of some iterative process.
- $\lambda(A)$: the set of eigenvalues of A.
- $\lambda \in \lambda(A)$: λ is an eigenvalue of A.

Existence and uniqueness

A linear equation system $Ax = b$ has a unique solution given by:

$$x = A^{-1}b$$

if and only if:

- A is non-singular.
- A has full rank.
- b $\in$ range(A).
- $\lambda \neq 0 \quad \forall \lambda \in \lambda(A)$
- (other equivalent properties are omitted).

Symmetric positive definite

A symmetric positive definite, SPD, matrix has the following properties:

- Symmetry: $A = A^T$.

Symmetric positive definite

A symmetric positive definite, SPD, matrix has the following properties:

- Symmetry: $A = A^T$.
- $\lambda > 0 \quad \forall \lambda \in \lambda(A)$

Symmetric positive definite

A symmetric positive definite, SPD, matrix has the following properties:

- Symmetry: $A = A^T$.
- $\lambda > 0 \quad \forall \lambda \in \lambda(A)$
- For any $x \neq 0, \quad x^T A x > 0$

Symmetric positive definite

A symmetric positive definite, SPD, matrix has the following properties:

- Symmetry: $A = A^T$.
- $\lambda > 0 \quad \forall \lambda \in \lambda(A)$
- For any $x \neq 0, \quad x^T A x > 0$
- All SPD matrices has full rank.

The symmetric difference matrix is SPD

Short proof by Gershgorin circle theorem.

- A is symmetric.

The symmetric difference matrix is SPD

Short proof by Gershgorin circle theorem.

- A is symmetric.
- Since A is weakly diagonally dominant all eigenvalues lie in the circle(interval) around 4 with radius 4.

The symmetric difference matrix is SPD

Short proof by Gershgorin circle theorem.

- A is symmetric.
- Since A is weakly diagonally dominant all eigenvalues lie in the circle(interval) around 4 with radius 4.
- Due to the irreducible property, A is non-singular and thus this interval is half-open.

The symmetric difference matrix is SPD

Short proof by Gershgorin circle theorem.

- A is symmetric.
- Since A is weakly diagonally dominant all eigenvalues lie in the circle(interval) around 4 with radius 4.
- Due to the irreducible property, A is non-singular and thus this interval is half-open.
- All eigenvalues are in the half-open set (0,8].

The symmetric difference matrix is SPD

Short proof by Gershgorin circle theorem.

- A is symmetric.
- Since A is weakly diagonally dominant all eigenvalues lie in the circle(interval) around 4 with radius 4.
- Due to the irreducible property, A is non-singular and thus this interval is half-open.
- All eigenvalues are in the half-open set (0,8].
- no eigenvalue is 0.

The symmetric difference matrix is SPD

Short proof by Gershgorin circle theorem.

- A is symmetric.
- Since A is weakly diagonally dominant all eigenvalues lie in the circle(interval) around 4 with radius 4.
- Due to the irreducible property, A is non-singular and thus this interval is half-open.
- All eigenvalues are in the half-open set (0,8].
- no eigenvalue is 0.
- All eigenvalues are > 0.

Eigenvalues and condition number

The eigenvalues of A are given by:

$$\lambda_k = 4\sin^2\left(\frac{\pi i}{2(n+1)}\right) + 4\sin^2\left(\frac{\pi j}{2(n+1)}\right), \quad k = i + (j-1)n, \quad i,j = 1..n.$$

The condition number of A is:

$$\kappa(A) = \frac{\lambda_{max}}{\lambda_{min}} \approx \frac{8}{8\sin^2\left(\frac{\pi}{2(n+1)}\right)} \approx \frac{1}{\frac{\pi^2}{4(n+1)^2}} = \frac{4(n+1)^2}{\pi^2} = \mathcal{O}(n^2).$$

Here we used the small angles approximation of sin.

Lipschitz continuity

- The spectral radius of a matrix, $\rho(A)$, is given by: $\rho(A) = \max_{\lambda \in \lambda(A)} |\lambda|$.

Lipschitz continuity

- The spectral radius of a matrix, $\rho(A)$, is given by: $\rho(A) = \max_{\lambda \in \lambda(A)} |\lambda|$.
- A mapping $\phi : \mathbb{R}^m \to \mathbb{R}^m$ is said to be globally Lipschitz continuous if and only if:

$$||\phi(x) - \phi(y)|| \leq L||x - y|| \quad \forall x, y \in \mathbb{R}^m,$$

holds for some $L \in \mathbb{R}^+$. The smallest such L is called the Lipschitz constant of ϕ. If $L < 1$ then ϕ is said to be a contraction.

Lipschitz continuity

- The spectral radius of a matrix, $\rho(A)$, is given by: $\rho(A) = \max_{\lambda \in \lambda(A)} |\lambda|$.
- A mapping $\phi : \mathbb{R}^m \rightarrow \mathbb{R}^m$ is said to be globally Lipschitz continuous if and only if:

$$||\phi(x) - \phi(y)|| \leq L||x - y|| \quad \forall x, y \in \mathbb{R}^m,$$

holds for some $L \in \mathbb{R}^+$. The smallest such L is called the Lipschitz constant of ϕ. If $L < 1$ then ϕ is said to be a contraction.

- An affine mapping $\psi : \mathbb{R}^m \rightarrow \mathbb{R}^m$ of the form: $\psi(x) = Mx + c$ with $M \in \mathbb{R}^{m \times m}$ and $x, c \in \mathbb{R}^m$ is Lipschitz continuous with Lipschitz constant $L = \rho(M)$.

Banach fixed-point theorem

- Let ϕ be a Lipschitz continuous mapping and consider the fixed-point problem:

$$x^* = \phi(x^*), \quad \textit{for some } x^* \in \mathbb{R}^m.$$

Banach fixed-point theorem

- Let ϕ be a Lipschitz continuous mapping and consider the fixed-point problem:

$$x^* = \phi(x^*), \quad \textit{for some } x^* \in \mathbb{R}^m.$$

- A solution exists and is unique if and only if ϕ is a contraction.

Banach fixed-point theorem

- Let ϕ be a Lipschitz continuous mapping and consider the fixed-point problem:

$$x^* = \phi(x^*), \qquad \textit{for some } x^* \in \mathbb{R}^m.$$

- A solution exists and is unique if and only if ϕ is a contraction.
- The fixed-point problem can be solved by a process called fixed-point iteration, given by:

$$x^{(k+1)} = \phi(x^{(k)})$$

where $x^{(k)} \in \mathbb{R}^m$ denotes the k'th iterate, and is started by an initial guess $x^{(0)} \in \mathbb{R}^m$.

Banach fixed-point theorem

- Let ϕ be a Lipschitz continuous mapping and consider the fixed-point problem:

$$x^* = \phi(x^*), \qquad \textit{for some } x^* \in \mathbb{R}^m.$$

- A solution exists and is unique if and only if ϕ is a contraction.
- The fixed-point problem can be solved by a process called fixed-point iteration, given by:

$$x^{(k+1)} = \phi(x^{(k)})$$

  where $x^{(k)} \in \mathbb{R}^m$ denotes the k'th iterate, and is started by an initial guess $x^{(0)} \in \mathbb{R}^m$.
- The fixed-point iteration converges globally to a unique x^* independent of initial guess $x^{(0)}$.

$$x^* = \lim_{k \to \infty} x^{(k)}.$$

Direct methods

We want to solve $Ax = b$.

- First option: finding A^{-1} and multiply by b.

Direct methods

We want to solve $Ax = b$.

- First option: finding A^{-1} and multiply by b.
- This won't work for several reasons.

Direct methods

We want to solve $Ax = b$.

- First option: finding A^{-1} and multiply by b.
- This won't work for several reasons.
  - Finding A^{-1} needs more memory than we have and it takes too long.

Direct methods

We want to solve $Ax = b$.

- First option: finding A^{-1} and multiply by b.
- This won't work for several reasons.
  - Finding A^{-1} needs more memory than we have and it takes too long.
  - It suffers from instability and ill-conditioness.

Direct methods

We want to solve $Ax = b$.

- First option: finding A^{-1} and multiply by b.
- This won't work for several reasons.
  - Finding A^{-1} needs more memory than we have and it takes too long.
  - It suffers from instability and ill-conditioness.
- Second option: LU or Cholesky.

Direct methods

We want to solve $Ax = b$.

- First option: finding A^{-1} and multiply by b.
- This won't work for several reasons.
  - Finding A^{-1} needs more memory than we have and it takes too long.
  - It suffers from instability and ill-conditioness.
- Second option: LU or Cholesky.
- These won't work either for the same reasons above.

Iteration methods

Why iterate?

- No memory expensive factorizations.

Iteration methods

Why iterate?

- No memory expensive factorizations.
- Matrix-vector multiplication is fast for sparse matrices.

Iteration methods

Why iterate?

- No memory expensive factorizations.
- Matrix-vector multiplication is fast for sparse matrices.
- Avoid accuracy decay due to ill-conditioness.

Stationary linear iterative methods, SLIM

The core idea for SLIM methods is to formulate $Ax = b$ as a fixed-point problem and then solve it by fixed-pont iteration.
There are different ways to achieve this, all based on a splitting of the form: $A = A_1 + A_2$

- $Ax = b$
- $(A_1 + A_2)x = b$
- $A_1x = -A_2x + b$
- $x = -A_1^{-1}A_2x + A_1^{-1}b = Mx + c$

A_1 should be chosen such that it is easy to invert/solve with for example forward/backward substitution and it should be a good approximate inverse to A. M is called the iteration matrix for the method. Convergence of the method is guaranteed if $\rho(M) < 1$. SPD matrices has this property.

Jacobi

Consider splittings on the form $A = L + D + U$. If we choose $A_1 = D$, $A_2 = L + U$ we get the Jacobi method with iteration matrix:

$$M_{JAC} = -D^{-1}(L + U)$$

Gauss-Seidel

If we instead choose $A_1 = L + D$, $A_2 = U$ we get the Gauss-Seidel method with iteration matrix:

$$M_{GS} = -(D + L)^{-1} U$$

Successive over relaxation, SOR

Similar to Gauss-Seidel but we introduce a weight factor ω as an extra degree of freedom. The benefit will be apparent later. SOR with $\omega = 1$ is the Gauss-Seidel method. The iteration matrix for SOR is:

$$M_{SOR}(\omega) = -(D + \omega L)^{-1}(\omega U + (\omega - 1)D)$$

Convergence rate

The convergence rate of the SLIM methods are given by:

$$||x^{(k+1)} - x^{(k)}|| \leq \rho(M)^k ||x^{(1)} - x^{(0)}||$$

Thus smaller spectral radius is better.

Relation between spectral radii

Given $0 < \mu = \rho(M_{JAC}) < 1$ the formula for the spectral radius of $M_{SOR}(\omega)$ is given by:

$$\rho(M_{SOR}(\omega)) = \begin{cases} \frac{1}{4}(\omega\mu + \sqrt{\omega^2\mu^2 - 4(\omega - 1)})^2 & 0 < \omega \leq \omega_{opt}, \\ \omega - 1 & \omega_{opt} \leq \omega < 2. \end{cases}$$

With ω_{opt}:

$$\omega_{opt} = 1 + \left(\frac{\mu}{1 + \sqrt{1 - \mu^2}}\right)^2$$

This gives:

$$0 < \rho(M_{SOR}(\omega_{opt})) < \rho(M_{SOR}(1)) = \rho(M_{GS}) = \mu^2 < \rho(M_{JAC}) = \mu < 1$$

μ can be computed via this formula:

$$1 - \frac{\lambda_{min}}{4} = \rho(M_{JAC}) = \frac{\lambda_{max}}{4} - 1$$

Krylov subspace methods

Other iterative methods are GMRES and CG.

- These methods are not suited for solving systems with spread out eigenvalues.

Krylov subspace methods

Other iterative methods are GMRES and CG.

- These methods are not suited for solving systems with spread out eigenvalues.
- A remedy is to use a preconditioner.

Krylov subspace methods

Other iterative methods are GMRES and CG.

- These methods are not suited for solving systems with spread out eigenvalues.
- A remedy is to use a preconditioner.
- The SLIM-methods can be used as preconditioners.

Multi-grid methods

- Multi-grid methods requires some attention in design.

Multi-grid methods

- Multi-grid methods requires some attention in design.
- They require a full solve at the coarsest level.

Multi-grid methods

- Multi-grid methods requires some attention in design.
- They require a full solve at the coarsest level.
- This solve has to be done by some other method.

Multi-grid methods

- Multi-grid methods requires some attention in design.
- They require a full solve at the coarsest level.
- This solve has to be done by some other method.
- For example CG with SOR preconditioner.

Multi-grid methods

- Multi-grid methods requires some attention in design.
- They require a full solve at the coarsest level.
- This solve has to be done by some other method.
- For example CG with SOR preconditioner.
- Another option is to mimic the non-linear case, FASMG, and not do a full solve.

Multi-grid methods

- Multi-grid methods requires some attention in design.
- They require a full solve at the coarsest level.
- This solve has to be done by some other method.
- For example CG with SOR preconditioner.
- Another option is to mimic the non-linear case, FASMG, and not do a full solve.
- Instead of the full solve, do a number of iterations of a SLIM precoditioner.

LUND UNIVERSITY

Algorithm choice and test problem

The SLIM methods were chosen due to their secondary use case as preconditioners.

As a test problem for the experiments the following choices were made for simplicity.

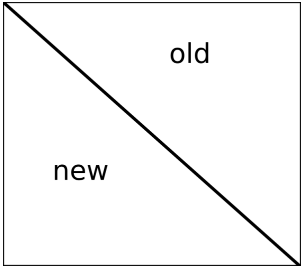
- $f(x,y) = \sin(\pi x) \cdot \sin(\pi y)$
- $\Omega = [0,1] \times [0,1] \subset \mathbb{R}^2$
- $g(x,y) \equiv 0$

Parallelism

We want a parallel implementation for increased performance. There are some pitfalls where the performance will be similar to a sequential implementation.
In this work parallelism through MPI was chosen over shared memory parallelization. Each process gets a partition of the full problem and need to share its data with its neighbouring processes. This communication was achieved with point-to-point communication.

The order matters

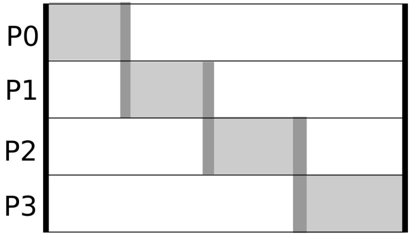
When we try to solve the equations with forward SOR we encounter a problem immediately.



In order to solve the first equation we need the newly updated values from the neighbouring process. This will cause a waiting problem.

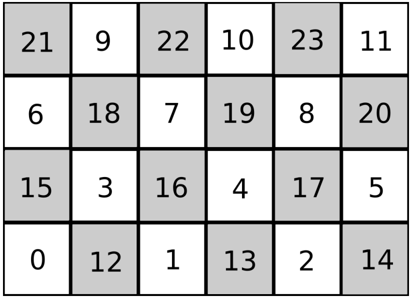
The order matters

The workflow this way is sequential, only one process has any work to do at a time.



Red-Black order

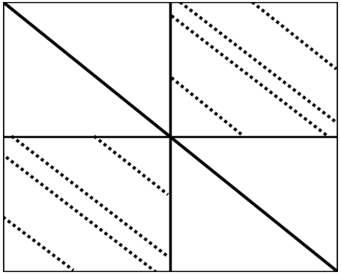
The remedy is to reorder the equations in the so-called Red-Black order.



21	9	22	10	23	11
6	18	7	19	8	20
15	3	16	4	17	5
0	12	1	13	2	14

New sparsity pattern

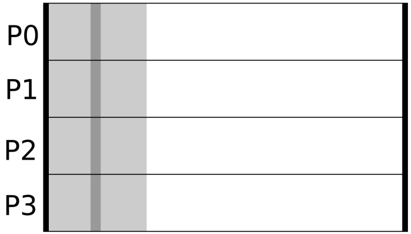
This gives the matrix a new sparsity pattern.



Now when we apply forward SOR every process has work to do since they only need old values for the first half of the equations.

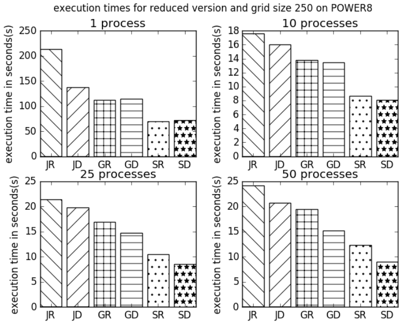
Red-Black SOR

The key insight is that we can solve the original system with a different order, the Red-Black order, instead of forward SOR. This way we avoid shuffling memory around. The workflow is now much improved.



Experiments

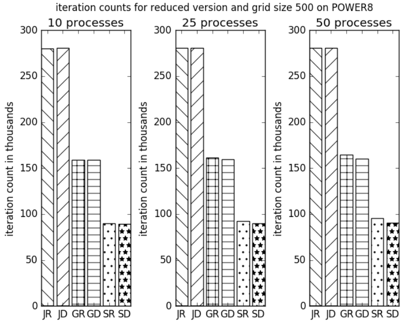
The experiments were performed on the POWER8 server at LTH. Some results:



Here we can see that going from 1 to 10 processes is a large improvement, whereas going to 25 or 50 brings no benefit.

Experiments continued

In another experiment we got:



Here we can see that the iteration count agrees somewhat with the theoretical estimate.

Conclusion and continued work

- SOR is faster than GS which is faster than Jacobi.
- More processes is faster for larger problems, up to a point.
- More processes for small problems gives no benefit. It can even reduce performance.
- Domain decomposition is slightly faster than by rows.
- Reduced and full versions are very similar in performance.

The next step is to implement these methods on a GPU, which i propose for continued work.

Questions

All source code and the report can be found at: https://gitlab.com/Drunte/public-sor

Questions?

The End

The End