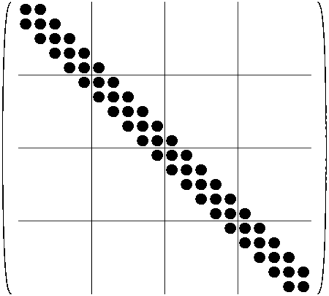
ulm university universität ulm

Scientific Computing

Parallele Algorithmen

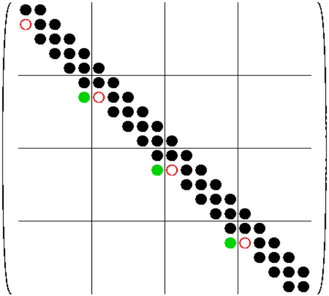
Prof. Dr. Stefan Funken, Prof. Dr. Alexander Keller,
Prof. Dr. Karsten Urban | 11. Januar 2007

How to solve a tridiagonal system?



Algorithm (Tridiagonal system)

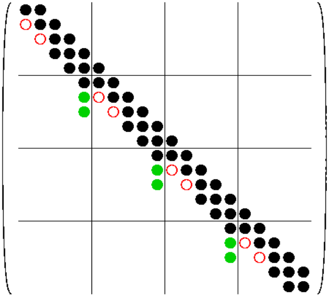
How to solve a tridiagonal system?



Algorithm (Tridiagonal system)

1. Eliminate in each diagonal block subdiagonal elements.

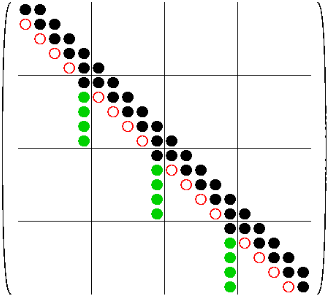
How to solve a tridiagonal system?



Algorithm (Tridiagonal system)

1. Eliminate in each diagonal block subdiagonal elements.

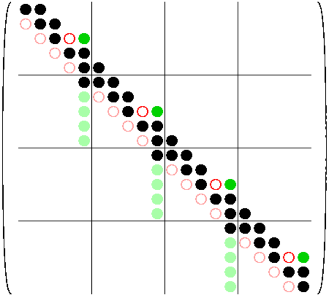
How to solve a tridiagonal system?



Algorithm (Tridiagonal system)

1. Eliminate in each diagonal block subdiagonal elements.

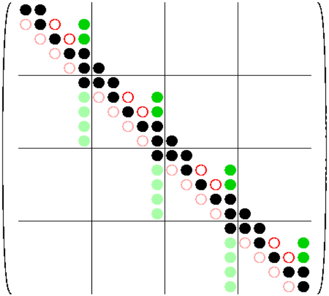
How to solve a tridiagonal system?



Algorithm (Tridiagonal system)

1. Eliminate in each diagonal block subdiagonal elements.
2. Eliminate in each diagonal block superdiagonal elements from third last row on.

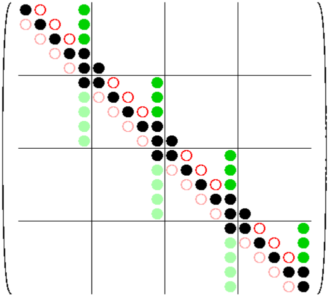
How to solve a tridiagonal system?



Algorithm (Tridiagonal system)

1. Eliminate in each diagonal block subdiagonal elements.
2. Eliminate in each diagonal block superdiagonal elements from third last row on.

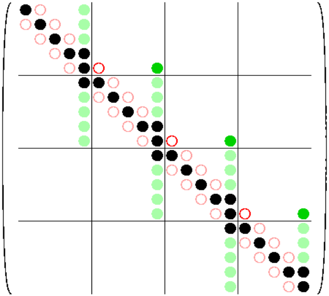
How to solve a tridiagonal system?



Algorithm (Tridiagonal system)

1. Eliminate in each diagonal block subdiagonal elements.
2. Eliminate in each diagonal block superdiagonal elements from third last row on.

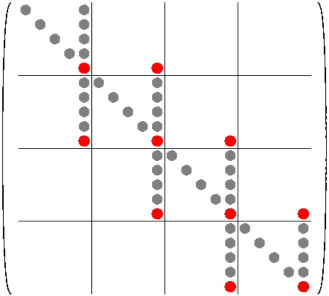
How to solve a tridiagonal system?



Algorithm (Tridiagonal system)

1. Eliminate in each diagonal block subdiagonal elements.
2. Eliminate in each diagonal block superdiagonal elements from third last row on.
3. Eliminate elements in superdiagonal blocks.

How to solve a tridiagonal system?

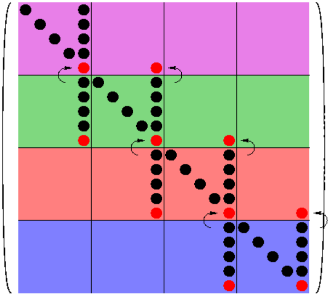


Algorithm (Tridiagonal system)

1. Eliminate in each diagonal block subdiagonal elements.
2. Eliminate in each diagonal block superdiagonal elements from third last row on.
3. Eliminate elements in superdiagonal blocks.

Results in a tridiagonal subsystem with unknowns x_5, x_{10}, x_{15}, x_{20}.

How to solve a tridiagonal system?

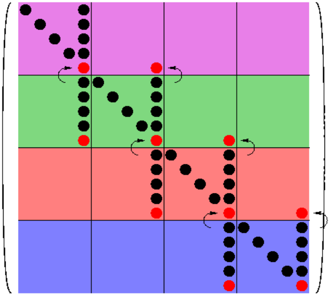


Algorithm (Tridiagonal system)

1. Eliminate in each diagonal block subdiagonal elements.
2. Eliminate in each diagonal block superdiagonal elements from third last row on.
3. Eliminate elements in superdiagonal blocks.

Results in a tridiagonal subsystem with unknowns x_5, x_{10}, x_{15}, x_{20}.

How to solve a tridiagonal system?



Algorithm (Tridiagonal system)

1. Eliminate in each diagonal block subdiagonal elements.
2. Eliminate in each diagonal block superdiagonal elements from third last row on.
3. Eliminate elements in superdiagonal blocks.

Results in a tridiagonal subsystem with unknowns x_5, x_{10}, x_{15}, x_{20}.
If data are stored rowwise only one communication to neighbouring processor neccessary.

Iterative Solver

Steepest Descent

The steepest descent method minimizes a differentiable function F in direction of steepest descent.
Consider $F(x) := \frac{1}{2}x^T Ax - b^T x$ where A is symmetric and positiv definite.
Hence, $\nabla F = \frac{1}{2}(A + A^T)x - b = Ax - b$

Input: Initial guess x^0

$r^0 := b - Ax^0$

Iteration: $k = 0, 1, \ldots$

$x^{k+1} := x^k + \lambda_{opt}(x^k, r^k)\, r^k$ % Update x^k

$r^{k+1} := b - Ax^{k+1}$ % Compute residual

Iterative Solver

Steepest Descent

The steepest descent method minimizes a differentiable function F in direction of steepest descent.
Consider $F(x) := \frac{1}{2}x^TAx - b^Tx$ where A is symmetric and positiv definite.
Hence, $\nabla F = \frac{1}{2}(A + A^T)x - b = Ax - b$

Input: Initial guess x^0

$r^0 := b - Ax^0$

Iteration: $k = 0, 1, \ldots$

$x^{k+1} := x^k + \lambda_{opt}(x^k, r^k)\, r^k$ % Update x^k

$r^{k+1} := b - Ax^{k+1}$ % Compute residual

Using $r^{k+1} = b - Ax^{k+1}$

Iterative Solver

Steepest Descent

The steepest descent method minimizes a differentiable function F in direction of steepest descent.
Consider $F(x) := \frac{1}{2}x^T Ax - b^T x$ where A is symmetric and positiv definite.
Hence, $\nabla F = \frac{1}{2}(A + A^T)x - b = Ax - b$

Input: Initial guess x^0

$r^0 := b - Ax^0$

Iteration: $k = 0, 1, \ldots$

$x^{k+1} := x^k + \lambda_{opt}(x^k, r^k)\, r^k$ % Update x^k

$r^{k+1} := b - Ax^{k+1}$ % Compute residual

Using $r^{k+1} = b - Ax^{k+1} = b - A(x^k + \lambda_{opt}(x^k, r^k)\, r^k)$

Iterative Solver

Steepest Descent

The steepest descent method minimizes a differentiable function F in direction of steepest descent.
Consider $F(x) := \frac{1}{2}x^T Ax - b^T x$ where A is symmetric and positiv definite.
Hence, $\nabla F = \frac{1}{2}(A + A^T)x - b = Ax - b$

Input: Initial guess x^0

$r^0 := b - Ax^0$

Iteration: $k = 0, 1, \ldots$

$x^{k+1} := x^k + \lambda_{opt}(x^k, r^k)\, r^k$ % Update x^k

$r^{k+1} := b - Ax^{k+1}$ % Compute residual

Using $r^{k+1} = b - Ax^{k+1} = b - A(x^k + \lambda_{opt}(x^k, r^k)\, r^k) = r^k - \lambda_{opt}(x^k, r^k)\, Ar^k$ gets

Iterative Solver

Steepest Descent

The steepest descent method minimizes a differentiable function F in direction of steepest descent.
Consider $F(x) := \frac{1}{2}x^T Ax - b^T x$ where A is symmetric and positiv definite.
Hence, $\nabla F = \frac{1}{2}(A + A^T)x - b = Ax - b$

Input: Initial guess x^0

$r^0 := b - Ax^0$

Iteration: $k = 0, 1, \ldots$

$x^{k+1} := x^k + \lambda_{opt}(x^k, r^k)\, r^k$ % Update x^k

$r^{k+1} := r^k - \lambda_{opt}(x^k, r^k)\, Ar^k$ % Compute residual

Steepest Descent Method

Let $x, p \in \mathbb{R}^n$. What is the optimal $\lambda_{opt}(x, p)$ in steepest descent method:
Consider the following minimization problem:

$$f(\lambda) \stackrel{!}{=} \min \quad \text{with} \quad f(\lambda) := F(x + \lambda p)$$

Then, with $F(x) = \frac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle$ we get

$$f(\lambda) = F(x + \lambda p)$$

Steepest Descent Method

Let $x, p \in \mathbb{R}^n$. What is the optimal $\lambda_{opt}(x, p)$ in steepest descent method:
Consider the following minimization problem:

$$f(\lambda) \stackrel{!}{=} \min \quad \text{with} \quad f(\lambda) := F(x + \lambda p)$$

Then, with $F(x) = \frac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle$ we get

$$\begin{aligned} f(\lambda) &= F(x + \lambda p) \\ &= \frac{1}{2}\langle x + \lambda p, A(x + \lambda p)\rangle - \langle b, x + \lambda p\rangle \end{aligned}$$

Steepest Descent Method

Let $x, p \in \mathbb{R}^n$. What is the optimal $\lambda_{opt}(x, p)$ in steepest descent method:
Consider the following minimization problem:

$$f(\lambda) \stackrel{!}{=} \min \quad \text{with} \quad f(\lambda) := F(x + \lambda p)$$

Then, with $F(x) = \frac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle$ we get

$$\begin{aligned} f(\lambda) &= F(x + \lambda p) \\ &= \frac{1}{2}\langle x + \lambda p, A(x + \lambda p)\rangle - \langle b, x + \lambda p\rangle \\ &= \frac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle + \lambda\langle p, Ax - b\rangle + \frac{1}{2}\lambda^2\langle p, Ap\rangle \end{aligned}$$

Steepest Descent Method

Let $x, p \in \mathbb{R}^n$. What is the optimal $\lambda_{opt}(x,p)$ in steepest descent method:
Consider the following minimization problem:

$$f(\lambda) \stackrel{!}{=} \min \quad \text{with} \quad f(\lambda) := F(x + \lambda p)$$

Then, with $F(x) = \frac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle$ we get

$$\begin{aligned}
f(\lambda) &= F(x+\lambda p) \\
&= \frac{1}{2}\langle x+\lambda p, A(x+\lambda p)\rangle - \langle b, x+\lambda p\rangle \\
&= \frac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle + \lambda\langle p, Ax-b\rangle + \frac{1}{2}\lambda^2\langle p, Ap\rangle \\
&= F(x) + \lambda\langle p, Ax-b\rangle + \frac{1}{2}\lambda^2\langle p, Ap\rangle
\end{aligned}$$

Steepest Descent Method

Let $x, p \in \mathbb{R}^n$. What is the optimal $\lambda_{opt}(x, p)$ in steepest descent method:
Consider the following minimization problem:

$$f(\lambda) \stackrel{!}{=} \min \quad \text{with} \quad f(\lambda) := F(x + \lambda p)$$

Then, with $F(x) = \frac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle$ we get

$$f(\lambda) \;=\; F(x) + \lambda\langle p, Ax - b\rangle + \frac{1}{2}\lambda^2\langle p, Ap\rangle$$

Steepest Descent Method

Let $x, p \in \mathbb{R}^n$. What is the optimal $\lambda_{opt}(x, p)$ in steepest descent method:
Consider the following minimization problem:

$$f(\lambda) \stackrel{!}{=} \min \quad \text{with} \quad f(\lambda) := F(x + \lambda p)$$

Then, with $F(x) = \frac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle$ we get

$$f(\lambda) \;=\; F(x) + \lambda\langle p, Ax - b\rangle + \frac{1}{2}\lambda^2\langle p, Ap\rangle$$

If $p \neq 0$, $\langle p, Ap\rangle > 0$.

Steepest Descent Method

Let $x, p \in \mathbb{R}^n$. What is the optimal $\lambda_{opt}(x, p)$ in steepest descent method:
Consider the following minimization problem:

$$f(\lambda) \overset{!}{=} \min \quad \text{with} \quad f(\lambda) := F(x + \lambda p)$$

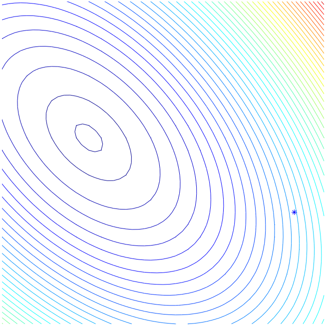
Then, with $F(x) = \frac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle$ we get

$$f(\lambda) \quad = \quad F(x) + \lambda\langle p, Ax - b\rangle + \frac{1}{2}\lambda^2\langle p, Ap\rangle$$

If $p \neq 0$, $\langle p, Ap\rangle > 0$.
Hence, from $0 \overset{!}{=} f'(\lambda) = \langle p, Ax - b\rangle + \lambda\langle p, Ap\rangle$ we obtain

$$\lambda_{opt}(x, p) = \frac{\langle p, b - Ax\rangle}{\langle p, Ap\rangle} \,.$$

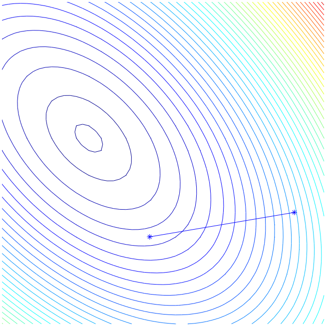
Numerical Example



2D Problem

- $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$
- $b = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$
- $x^0 = \begin{pmatrix} 8 \\ -3 \end{pmatrix}$
- 5 iterations

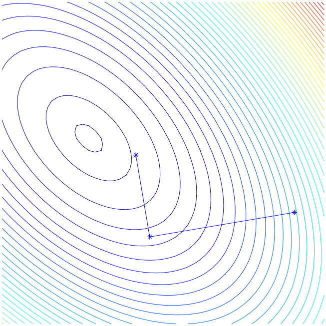
Numerical Example



2D Problem

- $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$
- $b = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$
- $x^0 = \begin{pmatrix} 8 \\ -3 \end{pmatrix}$
- 5 iterations

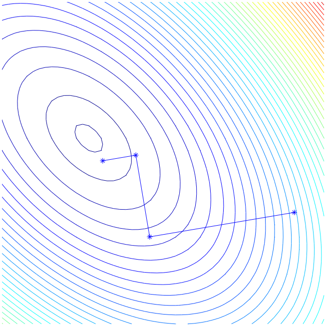
Numerical Example



2D Problem

- $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$
- $b = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$
- $x^0 = \begin{pmatrix} 8 \\ -3 \end{pmatrix}$
- 5 iterations

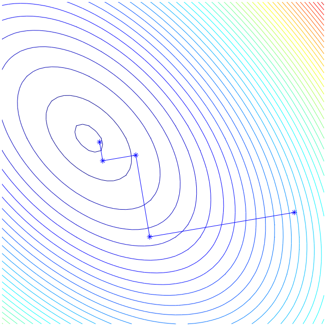
Numerical Example



2D Problem

- $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$
- $b = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$
- $x^0 = \begin{pmatrix} 8 \\ -3 \end{pmatrix}$
- 5 iterations

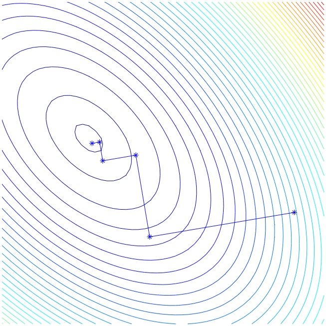
Numerical Example



2D Problem

- $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$
- $b = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$
- $x^0 = \begin{pmatrix} 8 \\ -3 \end{pmatrix}$
- 5 iterations

Numerical Example



2D Problem

- $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$
- $b = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$
- $x^0 = \begin{pmatrix} 8 \\ -3 \end{pmatrix}$
- 5 iterations

Iterative Solver

Steepest Descent

Input: Initial guess x^0

$r^0 := b - Ax^0$

Iteration: $k = 0, 1, \ldots$

$\lambda_{opt} := \frac{\langle r^k, r^k \rangle}{\langle r^k, Ar^k \rangle}$

$x^{k+1} := x^k + \lambda_{opt}\, r^k$

$r^{k+1} := r^k - \lambda_{opt}\, Ar^k$

2 matrix-vector-products, 2 inner products, and 2 saxpy's per iteration

Is it possible save one matrix-vector-product?

Iterative Solver

Steepest Descent

Input: Initial guess x^0

$r^0 := b - Ax^0$

Iteration: $k = 0, 1, \ldots$

$a^k := Ar^k$

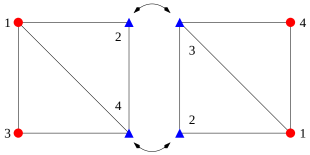
$\lambda_{opt} := \frac{\langle r^k, r^k \rangle}{\langle r^k, a^k \rangle}$

$x^{k+1} := x^k + \lambda_{opt}\, r^k$

$r^{k+1} := r^k - \lambda_{opt}\, a^k$

1 matrix-vector-products, 2 inner products, and 2 saxpy's per iteration

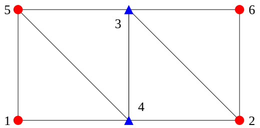
Numbering



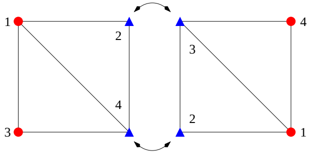
local numbering

How can vectors be given?

global numbering

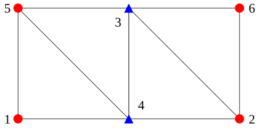


Numbering



local numbering

global numbering

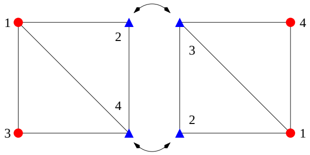


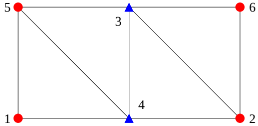
How can vectors be given?

- Full value at each node, e.g. given

$$u_\ell = (1,1,1,1)^T \quad u_r = (1,1,1,1)^T\,.$$

Numbering





How can vectors be given?

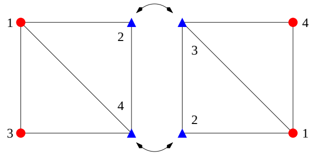
- Full value at each node, e.g. given

$$u_\ell = (1,1,1,1)^T \quad u_r = (1,1,1,1)^T\,.$$

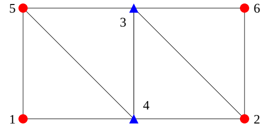
Using incidence matrices C_ℓ and C_r.

$$C_\ell = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad C_r = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Numbering



local numbering

global numbering


How can vectors be given?

- Full value at each node, e.g. given

$$u_\ell = (1,1,1,1)^T \quad u_r = (1,1,1,1)^T \,.$$

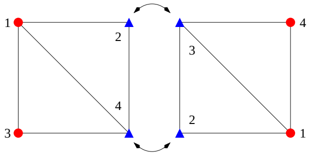
Using incidence matrices C_ℓ and C_r.

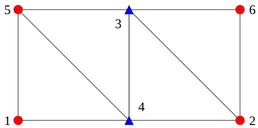
$$C_\ell = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad C_r = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Note

$$u_\ell : \quad \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Numbering





How can vectors be given?

- Full value at each node, e.g. given

$$u_\ell = (1,1,1,1)^T \quad u_r = (1,1,1,1)^T \,.$$

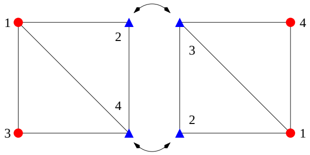
Hence

$$\begin{aligned} u &= C_\ell(1,1,1,1)^T + C_r(1,1,1,1)^T \\ &= (1,0,1,1,1,0)^T + (0,1,1,1,0,1)^T \\ &= (1,1,2,2,1,1)^T \neq (1,1,1,1,1,1)^T \end{aligned}$$

resp.

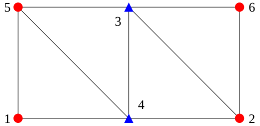
$$u = C_\ell u_\ell + C_r u_r$$

Numbering



local numbering

global numbering



How can vectors be given?

- Full value at each node be given.
- Value is given after assembling all data, e.g. given

$$u_\ell = (1, \frac{1}{2}, 1, \frac{1}{2})^T \quad u_r = (1, \frac{1}{2}, \frac{1}{2}, 1)^T$$

results in

$$\begin{aligned} u &= C_\ell u_\ell + C_r u_r \\ &= (1, 0, \frac{1}{2}, \frac{1}{2}, 1, 0)^T + (0, 1, \frac{1}{2}, \frac{1}{2}, 0, 1)^T \\ &= (1, 1, 1, 1, 1, 1)^T \end{aligned}$$

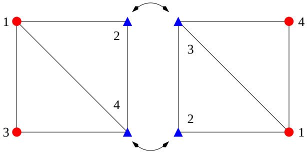
Types of Vectors

Two types of vectors, depending on the storage type:

type I: $\overline{u}$ is stored on P_k as restriction $\overline{u}_k = C_k \overline{u}$.
'Complete' value accessable on P_k.

type II: $\underline{r}$ is stored on P_k as $\underline{r}_k$, s.t.
$\underline{r} = \sum_{k=1}^{p} C_k^T \underline{r}_k$.
Nodes on the interface have only a part of the full value.

Numbering

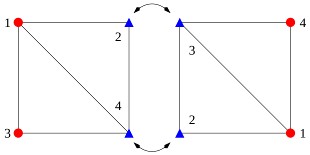


local numbering

Let matrices on both subdomains be given, for example:

$$A_\ell = \begin{pmatrix} 2 & 1 & 3 & -2 \\ -3 & 4 & -7 & 3 \\ 4 & 3 & 6 & 0 \\ 5 & -2 & 1 & 2 \end{pmatrix} \quad A_r = \begin{pmatrix} 0 & 2 & 1 & 0 \\ 1 & 3 & -7 & 2 \\ -2 & -9 & 4 & 0 \\ 3 & 7 & 1 & 5 \end{pmatrix}$$

Numbering



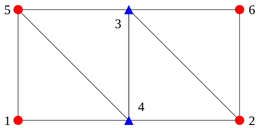
local numbering

Let matrices on both subdomains be given, for example:

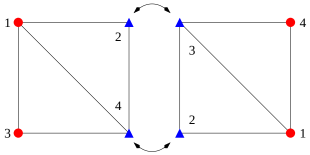
$$A_\ell = \begin{pmatrix} 2 & 1 & 3 & -2 \\ -3 & 4 & -7 & 3 \\ 4 & 3 & 6 & 0 \\ 5 & -2 & 1 & 2 \end{pmatrix} \quad A_r = \begin{pmatrix} 0 & 2 & 1 & 0 \\ 1 & 3 & -7 & 2 \\ -2 & -9 & 4 & 0 \\ 3 & 7 & 1 & 5 \end{pmatrix}$$

How to construct matrix A w.r.t global numbering from A_ℓ and A_r?

global numbering



Numbering



local numbering

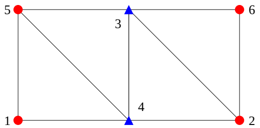
Let matrices on both subdomains be given, for example:

$$A_\ell = \begin{pmatrix} 2 & 1 & 3 & -2 \\ -3 & 4 & -7 & 3 \\ 4 & 3 & 6 & 0 \\ 5 & -2 & 1 & 2 \end{pmatrix} \quad A_r = \begin{pmatrix} 0 & 2 & 1 & 0 \\ 1 & 3 & -7 & 2 \\ -2 & -9 & 4 & 0 \\ 3 & 7 & 1 & 5 \end{pmatrix}$$

How to construct matrix A w.r.t global numbering from A_ℓ and A_r?

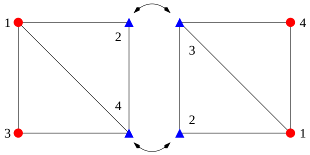
Use incidence matrices C_ℓ and C_r.

global numbering



$$C_\ell = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad C_r = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Numbering



local numbering

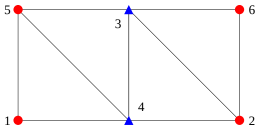
Let matrices on both subdomains be given, for example:

$$A_\ell = \begin{pmatrix} 2 & 1 & 3 & -2 \\ -3 & 4 & -7 & 3 \\ 4 & 3 & 6 & 0 \\ 5 & -2 & 1 & 2 \end{pmatrix} \quad A_r = \begin{pmatrix} 0 & 2 & 1 & 0 \\ 1 & 3 & -7 & 2 \\ -2 & -9 & 4 & 0 \\ 3 & 7 & 1 & 5 \end{pmatrix}$$

How to construct matrix A w.r.t global numbering from A_ℓ and A_r?

Use incidence matrices C_ℓ and C_r.

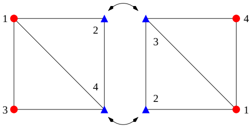
global numbering



$$C_\ell = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad C_r = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Now we get $A = C_\ell A_\ell C_\ell^T + C_r A_r C_r^T$.

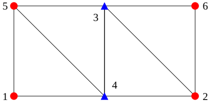
Numbering



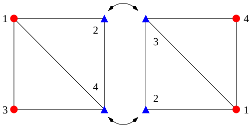
local numbering

$$A = C_\ell A_\ell C_\ell^T + C_r A_r C_r^T$$

global numbering



Numbering

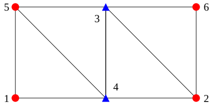


local numbering

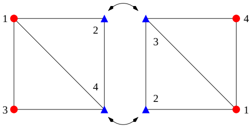
$$A = C_\ell A_\ell C_\ell^T + C_r A_r C_r^T$$

$$= \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 2 & 1 & 3 & -2 \\ -3 & 4 & -7 & 3 \\ 4 & 3 & 6 & 0 \\ 5 & -2 & 1 & 2 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} + \dots$$

global numbering



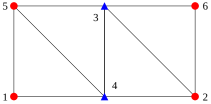
Numbering



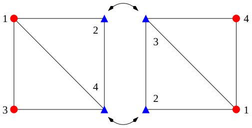
local numbering

$$
\begin{aligned}
A &= C_\ell A_\ell C_\ell^T + C_r A_r C_r^T \\
&= \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}
\begin{pmatrix} 2 & 1 & 3 & -2 \\ -3 & 4 & -7 & 3 \\ 4 & 3 & 6 & 0 \\ 5 & -2 & 1 & 2 \end{pmatrix}
\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} + \ldots \\
&= \begin{pmatrix} 6 & 0 & 3 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -7 & 0 & 4 & 3 & -3 & 0 \\ 1 & 0 & -2 & 2 & 5 & 0 \\ 3 & 0 & 1 & -2 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}
+ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & -2 & 4 & -9 & 0 & 0 \\ 0 & 1 & -7 & 3 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 1 & 7 & 0 & 5 \end{pmatrix}
\end{aligned}
$$

global numbering



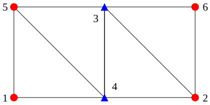
Numbering



local numbering

$$
\begin{aligned}
A &= C_\ell A_\ell C_\ell^T + C_r A_r C_r^T \\
&= \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}
\begin{pmatrix} 2 & 1 & 3 & -2 \\ -3 & 4 & -7 & 3 \\ 4 & 3 & 6 & 0 \\ 5 & -2 & 1 & 2 \end{pmatrix}
\begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} + \ldots \\
&= \begin{pmatrix} 6 & 0 & 3 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -7 & 0 & 4 & 3 & -3 & 0 \\ 1 & 0 & -2 & 2 & 5 & 0 \\ 3 & 0 & 1 & -2 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} +
\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & -2 & 4 & -9 & 0 & 0 \\ 0 & 1 & -7 & 3 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 1 & 7 & 0 & 5 \end{pmatrix} \\
&= \begin{pmatrix} 6 & 0 & 3 & 0 & 4 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 \\ -7 & -2 & 4+4 & -9+3 & -3 & 0 \\ 1 & 1 & -7-2 & 3+2 & 5 & 2 \\ 3 & 0 & 1 & -2 & 2 & 0 \\ 0 & 3 & 1 & 7 & 0 & 5 \end{pmatrix}
\end{aligned}
$$

global numbering



Types of Matrices

There are two types of matrices:

type I: 'Complete' (but not all) entries are accessable on P_k.

type II: The matrix is stored in a distrubuted manner similiar to type II.

$$A = \sum_{k=1}^{p} C_k A_k C_k^T$$

where A_k belongs to processor P_k, resp. to the subdomain Ω_i.

Converting Type

Obviously, addition, subtraction (and similiar operations) of vectors can be done without communication, if they are of the same type.

- Converting from type I to type II needs communication.
  Mapping is not unique, e.g.

$$\underline{u}_i = C_i \left(\sum_{k=1}^{p} C_k C_k^T \right)^{-1} C_k^T \overline{u}_k$$

- Converting from type II to type I needs communication.

$$\overline{r}_i = C_i \sum_{k=1}^{p} C_k^T \underline{r}_k$$

Inner Product

The inner product of two vectors $\overline{u}$, $\underline{r}$ of different type needs only one reduce-communication.

$$\langle \overline{u}, \underline{r} \rangle$$

Inner Product

The inner product of two vectors $\overline{u}$, $\underline{r}$ of different type needs only one reduce-communication.

$$\begin{aligned} & \langle \overline{u}, \underline{r} \rangle \\ = \; & \overline{u}^T \sum_{k=1}^{p} C_k^T \underline{r}_k \end{aligned}$$

Inner Product

The inner product of two vectors $\overline{u}$, $\underline{r}$ of different type needs only one reduce-communication.

$$
\begin{aligned}
&\langle \overline{u}, \underline{r} \rangle \\
= \quad & \overline{u}^T \sum_{k=1}^{p} C_k^T \underline{r}_k \\
= \quad & \sum_{k=1}^{p} \overline{u}^T C_k^T \underline{r}_k
\end{aligned}
$$

Inner Product

The inner product of two vectors $\overline{u}$, $\underline{r}$ of different type needs only one reduce-communication.

$$
\begin{aligned}
& \langle \overline{u}, \underline{r} \rangle \\
= \ & \overline{u}^T \sum_{k=1}^{p} C_k^T \underline{r}_k \\
= \ & \sum_{k=1}^{p} \overline{u}^T C_k^T \underline{r}_k \\
= \ & \sum_{k=1}^{p} \langle C_k \overline{u}, \underline{r}_k \rangle
\end{aligned}
$$

Inner Product

The inner product of two vectors $\overline{u}$, $\underline{r}$ of different type needs only one reduce-communication.

$$
\begin{aligned}
& \langle \overline{u}, \underline{r} \rangle \\
= \; & \overline{u}^T \sum_{k=1}^{p} C_k^T \underline{r}_k \\
= \; & \sum_{k=1}^{p} \overline{u}^T C_k^T \underline{r}_k \\
= \; & \sum_{k=1}^{p} \langle C_k \overline{u}, \underline{r}_k \rangle \\
= \; & \sum_{k=1}^{p} \langle \overline{u}_k, \underline{r}_k \rangle
\end{aligned}
$$

Matrix-Vector Multiplications

- type II - matrix $\times$ type I - vector
  result is a type II vector, no communication!!!
  Consider $A = \sum_{k=1}^{p} C_k A_k C_k^T$.

$$A\overline{u}$$

Matrix-Vector Multiplications

- type II - matrix $\times$ type I - vector
  result is a type II vector, no communication!!!
  Consider $A = \sum_{k=1}^{p} C_k A_k C_k^T$.

$$A\overline{u} = \sum_{k=1}^{p} C_k A_k C_k^T \overline{u}$$

Matrix-Vector Multiplications

- type II - matrix $\times$ type I - vector
  result is a type II vector, no communication!!!
  Consider $A = \sum_{k=1}^{p} C_k A_k C_k^T$.

$$A\overline{u} = \sum_{k=1}^{p} C_k A_k C_k^T \overline{u} = \sum_{k=1}^{p} C_k \underbrace{A_k \overline{u}_k}_{\underline{r}_k}$$

Matrix-Vector Multiplications

- type II - matrix × type I - vector
  result is a type II vector, no communication!!!
  Consider $A = \sum_{k=1}^{p} C_k A_k C_k^T$.

$$A\overline{u} = \sum_{k=1}^{p} C_k A_k C_k^T \overline{u} = \sum_{k=1}^{p} C_k \underbrace{A_k \overline{u}_k}_{\underline{r}_k} = \underline{r}$$

- type II - matrix × type II - vector
  type conversion neccessary, needs communication

Steepest Descent

Parallel Version

Input: Initial guess $\overline{x}^0$

$\underline{r}^0 := \underline{b} - A\overline{x}^0$

$\overline{w}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{r}^0$

Iteration: $k = 0, 1, \ldots$

$\underline{a}^k := A\overline{w}^k$

$\lambda := \frac{\langle \overline{w}^k, \underline{r}^k \rangle}{\langle \overline{w}^k, \underline{a}^k \rangle}$

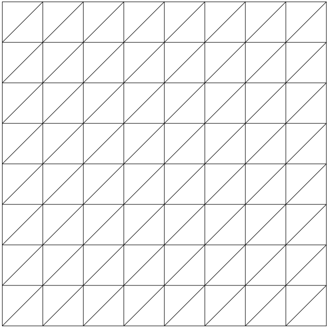
$\overline{x}^{k+1} := \overline{x}^k + \lambda\, \overline{w}^k$

$\underline{r}^{k+1} := \underline{r}^k - \lambda\, \underline{a}^k$

$\overline{w}^k := \sum_{\ell=1}^{p} C_\ell^T \underline{r}^k$

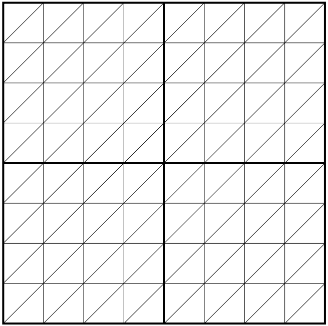
Only two allreduce-communications and one vector accumulation per iteration necessary!

Non-overlapping Subdomains



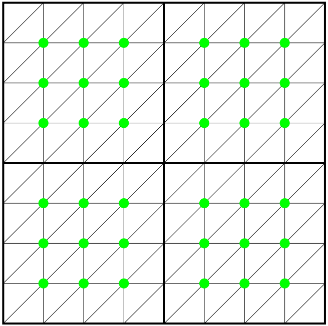
Different Indizes

Non-overlapping Subdomains



Different Indizes

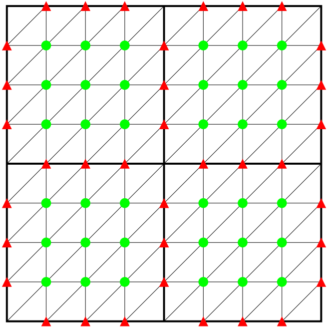
Non-overlapping Subdomains



Different Indizes

1. **I** nodes in interior of subdomains $[N_I = \sum_{j=1}^{p} N_{I,j}]$.

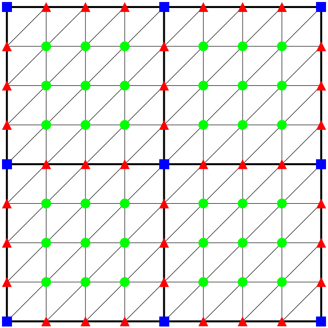
Non-overlapping Subdomains



Different Indizes

1. **I** nodes in interior of subdomains $[N_I = \sum_{j=1}^{p} N_{I,j}]$.
2. **E** nodes in interior of subdomains-edges $[N_E = \sum_{j=1}^{n_e} N_{E,j}]$. ($n_e$ number of subdomain-edges)

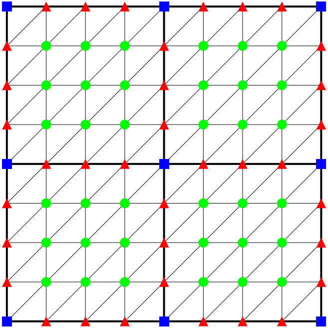
Non-overlapping Subdomains



Different Indizes

1. **I** nodes in interior of subdomains $[N_I = \sum_{j=1}^{p} N_{I,j}]$.
2. **E** nodes in interior of subdomains-edges $[N_E = \sum_{j=1}^{n_e} N_{E,j}]$. ($n_e$ number of subdomain-edges)
3. **V** crosspoints, i.e. endpoints of subdomain-edges $[N_V]$

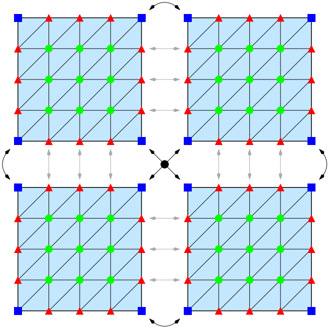
Non-overlapping Subdomains



Different Indizes

1. **I** nodes in interior of subdomains $[N_I = \sum_{j=1}^{p} N_{I,j}]$.
2. **E** nodes in interior of subdomains-edges $[N_E = \sum_{j=1}^{n_e} N_{E,j}]$. ($n_e$ number of subdomain-edges)
3. **V** crosspoints, i.e. endpoints of subdomain-edges $[N_V]$
4. **E** and **V** are often denoted as coupling nodes with index **C** $[N_C = N_E + N_V]$

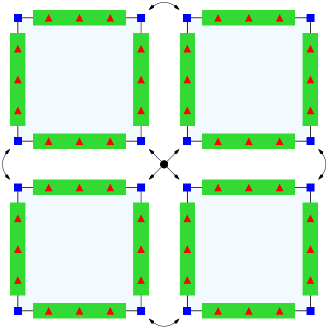
Non-overlapping Subdomains



Communication

1. Communication only neccessary for nodes on the coupling boundaries.

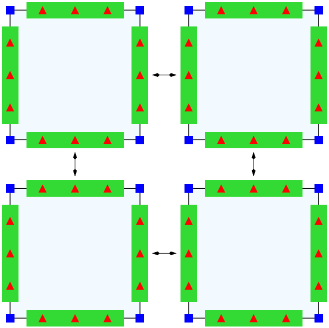
Non-overlapping Subdomains



Communication

1. Communication only neccessary for nodes on the coupling boundaries.
2. Global communication for crosspoints.

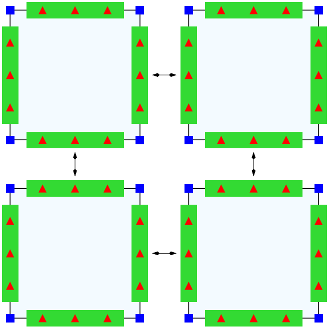
Non-overlapping Subdomains



Communication

1. Communication only neccessary for nodes on the coupling boundaries.
2. Global communication for crosspoints.
3. Only communication to the neighbouring subdomain for edge-nodes.

Non-overlapping Subdomains

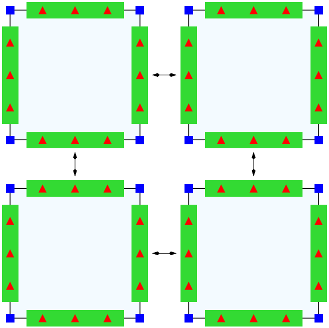


Communication

1. Communication only neccessary for nodes on the coupling boundaries.
2. Global communication for crosspoints.
3. Only communication to the neighbouring subdomain for edge-nodes.
4. Not all nodes have to be 'touched' for a vector accumulation

$$\overline{w} := \sum_{\ell=1}^{p} C_\ell^T \underline{r}$$

Non-overlapping Subdomains



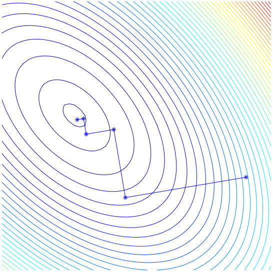
Communication

1. Communication only neccessary for nodes on the coupling boundaries.
2. Global communication for crosspoints.
3. Only communication to the neighbouring subdomain for edge-nodes.
4. Not all nodes have to be 'touched' for a vector accumulation

$$\overline{w} := \sum_{\ell=1}^{p} C_\ell^T \underline{r}$$

5. Split into communication between neighbouring subdomains and one global communication for all crosspoints.

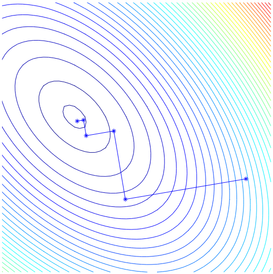
Numerical Example



Notice the following properties of the algorithm

$$r^m \perp r^{m+1} = r^m - \lambda_{opt}(x^m, r^m)\, Ar^m = r^m - \frac{\langle r^m, b - Ax^m \rangle}{\langle r^m, Ar^m \rangle} Ar^m$$

Numerical Example



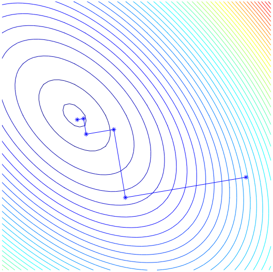
Notice the following properties of the algorithm

$$r^m \perp r^{m+1} = r^m - \lambda_{opt}(x^m, r^m)\, Ar^m = r^m - \frac{\langle r^m, b - Ax^m \rangle}{\langle r^m, Ar^m \rangle} Ar^m$$

resp.

$$\langle r^m, r^{m+1} \rangle$$

Numerical Example



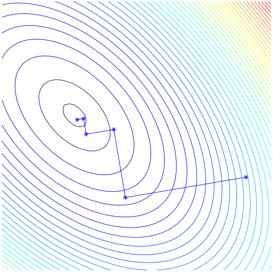
Notice the following properties of the algorithm

$$r^m \perp r^{m+1} = r^m - \lambda_{opt}(x^m, r^m)\, Ar^m = r^m - \frac{\langle r^m, b - Ax^m \rangle}{\langle r^m, Ar^m \rangle} Ar^m$$

resp.

$$\langle r^m, r^{m+1} \rangle = \langle r^m, r^m \rangle - \frac{\langle r^m, b - Ax^m \rangle}{\langle r^m, Ar^m \rangle} \langle r^m, Ar^m \rangle = 0$$

Numerical Example



Notice the following properties of the algorithm

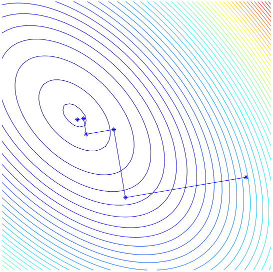
$$r^m \perp r^{m+1} = r^m - \lambda_{opt}(x^m, r^m)\, Ar^m = r^m - \frac{\langle r^m, b - Ax^m\rangle}{\langle r^m, Ar^m\rangle} Ar^m$$

resp.

$$\langle r^m, r^{m+1}\rangle = \langle r^m, r^m\rangle - \frac{\langle r^m, b - Ax^m\rangle}{\langle r^m, Ar^m\rangle}\langle r^m, Ar^m\rangle = 0$$

but not $r^m \perp r^{m+2}$.

Numerical Example



Notice the following properties of the algorithm

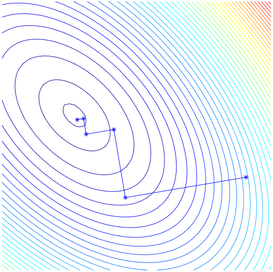
$$r^m \perp r^{m+1} = r^m - \lambda_{opt}(x^m, r^m)\, Ar^m = r^m - \frac{\langle r^m, b - Ax^m \rangle}{\langle r^m, Ar^m \rangle} Ar^m$$

resp.

$$\langle r^m, r^{m+1} \rangle = \langle r^m, r^m \rangle - \frac{\langle r^m, b - Ax^m \rangle}{\langle r^m, Ar^m \rangle} \langle r^m, Ar^m \rangle = 0$$

but not $r^m \perp r^{m+2}$. We loose all our information!!!
There exists a better algorithm for symmetric and positive definite matrices, as they arise in the finite element method!!!

Numerical Example



Notice the following properties of the algorithm

$$r^m \perp r^{m+1} = r^m - \lambda_{opt}(x^m, r^m)\, Ar^m = r^m - \frac{\langle r^m, b - Ax^m \rangle}{\langle r^m, Ar^m \rangle} Ar^m$$

resp.

$$\langle r^m, r^{m+1} \rangle = \langle r^m, r^m \rangle - \frac{\langle r^m, b - Ax^m \rangle}{\langle r^m, Ar^m \rangle} \langle r^m, Ar^m \rangle = 0$$

but not $r^m \perp r^{m+2}$. We loose all our information!!!
There exists a better algorithm for symmetric and positive definite matrices, as they arise in the finite element method!!! The CG-algorithm.

Preconditioned Conjugate Gradient Method

Solve $Ax = b$ (A, W sym, + def), W^{-1} 'easy' to compute, s.t. $W^{-1}A \approx I$ (e.g. $W^{-1} = I$, $W^{-1} = k$-iterations of Jacobi/Gauss-Seidel)

Input: Initial guess x^0

$r^0 := b - Ax^0$

$p^0 := W^{-1} r^0$

$\sigma_0 := \langle p^0, r^0 \rangle$

Iteration: $k = 0, 1, \ldots$ (as long as $k < n$, $r^k \neq 0$)

$a^k := Ap^k$

$\lambda_{opt} := \frac{\sigma_k}{\langle a^k, p^k \rangle}$

$x^{k+1} := x^k + \lambda_{opt}\, r^k$

$r^{k+1} := r^k - \lambda_{opt}\, a^k$

$q^{k+1} := W^{-1} r^{k+1}$

$\sigma_{k+1} := \langle q^{k+1}, r^{k+1} \rangle$

$p^{m+1} := q^{m+1} + \frac{\sigma_{k+1}}{\sigma_k} p^k$

Parallel Preconditioned Conjugate Gradient Method

Input: Initial guess x^0

$r^0 := b - Ax^0$

$p^0 := W^{-1} r^0$

$\sigma_0 := \langle p^0, r^0 \rangle$

Iteration: $k = 0, 1, \ldots$ (as long as $k < n$, $r^k \neq 0$)

$a^k := Ap^k, \quad \lambda_{opt} := \frac{\sigma_k}{\langle a^k, p^k \rangle}$

$x^{k+1} := x^k + \lambda_{opt}\, p^k$

$r^{k+1} := r^k - \lambda_{opt}\, a^k$

$q^{k+1} := W^{-1} r^{k+1}, \quad \sigma_{k+1} := \langle q^{k+1}, r^{k+1} \rangle$

$p^{k+1} := q^{k+1} + \frac{\sigma_{k+1}}{\sigma_k} p^k$

Parallel Preconditioned Conjugate Gradient Method

Input: Initial guess $\overline{x}^0$

$r^0 := b - Ax^0$

$p^0 := W^{-1} r^0$

$\sigma_0 := \langle p^0, r^0 \rangle$

Iteration: $k = 0, 1, \ldots$ (as long as $k < n$, $r^k \neq 0$)

$a^k := Ap^k, \quad \lambda_{opt} := \frac{\sigma_k}{\langle a^k, p^k \rangle}$

$x^{k+1} := x^k + \lambda_{opt}\, p^k$

$r^{k+1} := r^k - \lambda_{opt}\, a^k$

$q^{k+1} := W^{-1} r^{k+1}, \quad \sigma_{k+1} := \langle q^{k+1}, r^{k+1} \rangle$

$p^{k+1} := q^{k+1} + \frac{\sigma_{k+1}}{\sigma_k} p^k$

Parallel Preconditioned Conjugate Gradient Method

Input: Initial guess $\overline{x}^0$

$\underline{r}^0 := \underline{b} - A\overline{x}^0$

$p^0 := W^{-1} r^0$

$\sigma_0 := \langle p^0, r^0 \rangle$

Iteration: $k = 0, 1, \ldots$ (as long as $k < n$, $r^k \neq 0$)

$a^k := Ap^k, \quad \lambda_{opt} := \frac{\sigma_k}{\langle a^k, p^k \rangle}$

$x^{k+1} := x^k + \lambda_{opt}\, p^k$

$r^{k+1} := r^k - \lambda_{opt}\, a^k$

$q^{k+1} := W^{-1} r^{k+1}, \quad \sigma_{k+1} := \langle q^{k+1}, r^{k+1} \rangle$

$p^{k+1} := q^{k+1} + \frac{\sigma_{k+1}}{\sigma_k} p^k$

Parallel Preconditioned Conjugate Gradient Method

Input: Initial guess $\overline{x}^0$

$\underline{r}^0 := \underline{b} - A\overline{x}^0$

$\overline{w}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{r}^0$

$p^0 := W^{-1} r^0$

$\sigma_0 := \langle p^0, r^0 \rangle$

Iteration: $k = 0, 1, \ldots$ (as long as $k < n$, $r^k \neq 0$)

$a^k := Ap^k, \quad \lambda_{opt} := \frac{\sigma_k}{\langle a^k, p^k \rangle}$

$x^{k+1} := x^k + \lambda_{opt}\, p^k$

$r^{k+1} := r^k - \lambda_{opt}\, a^k$

$q^{k+1} := W^{-1} r^{k+1}, \quad \sigma_{k+1} := \langle q^{k+1}, r^{k+1} \rangle$

$p^{k+1} := q^{k+1} + \frac{\sigma_{k+1}}{\sigma_k} p^k$

Parallel Preconditioned Conjugate Gradient Method

Input: Initial guess $\overline{x}^0$

$\underline{r}^0 := \underline{b} - A\overline{x}^0$

$\overline{w}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{r}^0$

$\underline{p}^0 := W^{-1}\overline{w}^0$

$\sigma_0 := \langle p^0, r^0 \rangle$

Iteration: $k = 0, 1, \ldots$ (as long as $k < n$, $r^k \neq 0$)

$a^k := Ap^k, \quad \lambda_{opt} := \frac{\sigma_k}{\langle a^k, p^k \rangle}$

$x^{k+1} := x^k + \lambda_{opt}\, p^k$

$r^{k+1} := r^k - \lambda_{opt}\, a^k$

$q^{k+1} := W^{-1} r^{k+1}, \quad \sigma_{k+1} := \langle q^{k+1}, r^{k+1} \rangle$

$p^{k+1} := q^{k+1} + \frac{\sigma_{k+1}}{\sigma_k} p^k$

Parallel Preconditioned Conjugate Gradient Method

Input: Initial guess $\overline{x}^0$

$\underline{r}^0 := \underline{b} - A\overline{x}^0$

$\overline{w}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{r}^0$

$\underline{p}^0 := W^{-1}\overline{w}^0$

$\overline{s}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{p}^0$

$\sigma_0 := \langle \overline{w}^0, \underline{p}^0 \rangle$

Iteration: $k = 0, 1, \ldots$ (as long as $k < n$, $r^k \neq 0$)

$a^k := Ap^k, \quad \lambda_{opt} := \frac{\sigma_k}{\langle a^k, p^k \rangle}$

$x^{k+1} := x^k + \lambda_{opt}\, p^k$

$r^{k+1} := r^k - \lambda_{opt}\, a^k$

$q^{k+1} := W^{-1} r^{k+1}, \quad \sigma_{k+1} := \langle q^{k+1}, r^{k+1} \rangle$

$p^{k+1} := q^{k+1} + \frac{\sigma_{k+1}}{\sigma_k} p^k$

Parallel Preconditioned Conjugate Gradient Method

Input: Initial guess $\overline{x}^0$

$\underline{r}^0 := \underline{b} - A\overline{x}^0$

$\overline{w}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{r}^0$

$\underline{p}^0 := W^{-1}\overline{w}^0$

$\overline{s}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{p}^0$

$\sigma_0 := \langle \overline{w}^0, \underline{p}^0 \rangle$

Iteration: $k = 0, 1, \ldots$ (as long as $k < n$, $r^k \neq 0$)

$\underline{a}^k := A\overline{s}^k$, $\lambda_{opt} := \frac{\sigma_k}{\langle a^k, p^k \rangle}$

$x^{k+1} := x^k + \lambda_{opt}\, p^k$

$r^{k+1} := r^k - \lambda_{opt}\, a^k$

$q^{k+1} := W^{-1} r^{k+1}$, $\sigma_{k+1} := \langle q^{k+1}, r^{k+1} \rangle$

$p^{k+1} := q^{k+1} + \frac{\sigma_{k+1}}{\sigma_k} p^k$

Parallel Preconditioned Conjugate Gradient Method

Input: Initial guess $\overline{x}^0$

$\underline{r}^0 := \underline{b} - A\overline{x}^0$

$\overline{w}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{r}^0$

$\underline{p}^0 := W^{-1}\overline{w}^0$

$\overline{s}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{p}^0$

$\sigma_0 := \langle \overline{w}^0, \underline{p}^0 \rangle$

Iteration: $k = 0, 1, \ldots$ (as long as $k < n$, $r^k \neq 0$)

$\underline{a}^k := A\overline{s}^k$, $\lambda_{opt} := \frac{\sigma_k}{\langle \underline{a}^k, \overline{s}^k \rangle}$

$x^{k+1} := x^k + \lambda_{opt}\, p^k$

$r^{k+1} := r^k - \lambda_{opt}\, a^k$

$q^{k+1} := W^{-1} r^{k+1}$, $\sigma_{k+1} := \langle q^{k+1}, r^{k+1} \rangle$

$p^{k+1} := q^{k+1} + \frac{\sigma_{k+1}}{\sigma_k} p^k$

Parallel Preconditioned Conjugate Gradient Method

Input: Initial guess $\overline{x}^0$

$\underline{r}^0 := \underline{b} - A\overline{x}^0$

$\overline{w}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{r}^0$

$\underline{p}^0 := W^{-1}\overline{w}^0$

$\overline{s}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{p}^0$

$\sigma_0 := \langle \overline{w}^0, \underline{p}^0 \rangle$

Iteration: $k = 0, 1, \ldots$ (as long as $k < n$, $r^k \neq 0$)

$\underline{a}^k := A\overline{s}^k$, $\lambda_{opt} := \frac{\sigma_k}{\langle \underline{a}^k, \overline{s}^k \rangle}$

$\overline{x}^{k+1} := \overline{x}^k + \lambda_{opt}\, \overline{s}^k$

$r^{k+1} := r^k - \lambda_{opt}\, a^k$

$q^{k+1} := W^{-1} r^{k+1}$, $\sigma_{k+1} := \langle q^{k+1}, r^{k+1} \rangle$

$p^{k+1} := q^{k+1} + \frac{\sigma_{k+1}}{\sigma_k} p^k$

Parallel Preconditioned Conjugate Gradient Method

Input: Initial guess $\overline{x}^0$

$\underline{r}^0 := \underline{b} - A\overline{x}^0$

$\overline{w}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{r}^0$

$\underline{p}^0 := W^{-1}\overline{w}^0$

$\overline{s}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{p}^0$

$\sigma_0 := \langle \overline{w}^0, \underline{p}^0 \rangle$

Iteration: $k = 0, 1, \ldots$ (as long as $k < n$, $r^k \neq 0$)

$\underline{a}^k := A\overline{s}^k, \quad \lambda_{opt} := \frac{\sigma_k}{\langle \underline{a}^k, \overline{s}^k \rangle}$

$\overline{x}^{k+1} := \overline{x}^k + \lambda_{opt}\, \overline{s}^k$

$\underline{r}^{k+1} := \underline{r}^k - \lambda_{opt}\, \underline{a}^k$

$q^{k+1} := W^{-1} r^{k+1}, \quad \sigma_{k+1} := \langle q^{k+1}, r^{k+1} \rangle$

$p^{k+1} := q^{k+1} + \frac{\sigma_{k+1}}{\sigma_k} p^k$

Parallel Preconditioned Conjugate Gradient Method

Input: Initial guess $\overline{x}^0$

$\underline{r}^0 := \underline{b} - A\overline{x}^0$

$\overline{w}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{r}^0$

$\underline{p}^0 := W^{-1}\overline{w}^0$

$\overline{s}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{p}^0$

$\sigma_0 := \langle \overline{w}^0, \underline{p}^0 \rangle$

Iteration: $k = 0, 1, \ldots$ (as long as $k < n$, $r^k \neq 0$)

$\underline{a}^k := A\overline{s}^k$, $\lambda_{opt} := \frac{\sigma_k}{\langle \underline{a}^k, \overline{s}^k \rangle}$

$\overline{x}^{k+1} := \overline{x}^k + \lambda_{opt}\,\overline{s}^k$

$\underline{r}^{k+1} := \underline{r}^k - \lambda_{opt}\,\underline{a}^k$

$\overline{w}^{k+1} := \sum_{\ell=1}^{p} C_\ell^T \underline{r}^{k+1}$

$q^{k+1} := W^{-1} r^{k+1}$, $\sigma_{k+1} := \langle q^{k+1}, r^{k+1} \rangle$

$p^{k+1} := q^{k+1} + \frac{\sigma_{k+1}}{\sigma_k} p^k$

Parallel Preconditioned Conjugate Gradient Method

Input: Initial guess $\overline{x}^0$

$\underline{r}^0 := \underline{b} - A\overline{x}^0$

$\overline{w}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{r}^0$

$\underline{p}^0 := W^{-1}\overline{w}^0$

$\overline{s}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{p}^0$

$\sigma_0 := \langle \overline{w}^0, \underline{p}^0 \rangle$

Iteration: $k = 0, 1, \ldots$ (as long as $k < n$, $r^k \neq 0$)

$\underline{a}^k := A\overline{s}^k, \quad \lambda_{opt} := \frac{\sigma_k}{\langle \underline{a}^k, \overline{s}^k \rangle}$

$\overline{x}^{k+1} := \overline{x}^k + \lambda_{opt}\, \overline{s}^k$

$\underline{r}^{k+1} := \underline{r}^k - \lambda_{opt}\, \underline{a}^k$

$\overline{w}^{k+1} := \sum_{\ell=1}^{p} C_\ell^T \underline{r}^{k+1}$

$\underline{q}^{k+1} := W^{-1}\overline{w}^{k+1}, \quad \sigma_{k+1} := \langle q^{k+1}, r^{k+1} \rangle$

$p^{k+1} := q^{k+1} + \frac{\sigma_{k+1}}{\sigma_k} p^k$

Parallel Preconditioned Conjugate Gradient Method

Input: Initial guess $\overline{x}^0$

$\underline{r}^0 := \underline{b} - A\overline{x}^0$

$\overline{w}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{r}^0$

$\underline{p}^0 := W^{-1}\overline{w}^0$

$\overline{s}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{p}^0$

$\sigma_0 := \langle \overline{w}^0, \underline{p}^0 \rangle$

Iteration: $k = 0, 1, \ldots$ (as long as $k < n$, $r^k \neq 0$)

$\underline{a}^k := A\overline{s}^k, \quad \lambda_{opt} := \frac{\sigma_k}{\langle \underline{a}^k, \overline{s}^k \rangle}$

$\overline{x}^{k+1} := \overline{x}^k + \lambda_{opt}\, \overline{s}^k$

$\underline{r}^{k+1} := \underline{r}^k - \lambda_{opt}\, \underline{a}^k$

$\overline{w}^{k+1} := \sum_{\ell=1}^{p} C_\ell^T \underline{r}^{k+1}$

$\underline{q}^{k+1} := W^{-1}\overline{w}^{k+1}, \quad \sigma_{k+1} := \langle \underline{q}^{k+1}, \overline{w}^{k+1} \rangle$

$p^{k+1} := q^{k+1} + \frac{\sigma_{k+1}}{\sigma_k} p^k$

Parallel Preconditioned Conjugate Gradient Method

Input: Initial guess $\overline{x}^0$

$\underline{r}^0 := \underline{b} - A\overline{x}^0$

$\overline{w}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{r}^0$

$\underline{p}^0 := W^{-1}\overline{w}^0$

$\overline{s}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{p}^0$

$\sigma_0 := \langle \overline{w}^0, \underline{p}^0 \rangle$

Iteration: $k = 0, 1, \ldots$ (as long as $k < n$, $r^k \neq 0$)

$\underline{a}^k := A\overline{s}^k, \quad \lambda_{opt} := \frac{\sigma_k}{\langle \underline{a}^k, \overline{s}^k \rangle}$

$\overline{x}^{k+1} := \overline{x}^k + \lambda_{opt}\, \overline{s}^k$

$\underline{r}^{k+1} := \underline{r}^k - \lambda_{opt}\, \underline{a}^k$

$\overline{w}^{k+1} := \sum_{\ell=1}^{p} C_\ell^T \underline{r}^{k+1}$

$\underline{q}^{k+1} := W^{-1}\overline{w}^{k+1}, \quad \sigma_{k+1} := \langle \underline{q}^{k+1}, \overline{w}^{k+1} \rangle$

$\underline{p}^{k+1} := \underline{q}^{k+1} + \frac{\sigma_{k+1}}{\sigma_k} \underline{p}^k$

Parallel Preconditioned Conjugate Gradient Method

Input: Initial guess $\overline{x}^0$

$\underline{r}^0 := \underline{b} - A\overline{x}^0$

$\overline{w}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{r}^0$

$\underline{p}^0 := W^{-1}\overline{w}^0$

$\overline{s}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{p}^0$

$\sigma_0 := \langle \overline{w}^0, \underline{p}^0 \rangle$

Iteration: $k = 0, 1, \dots$ (as long as $k < n$, $r^k \neq 0$)

$\underline{a}^k := A\overline{s}^k, \quad \lambda_{opt} := \frac{\sigma_k}{\langle \underline{a}^k, \overline{s}^k \rangle}$

$\overline{x}^{k+1} := \overline{x}^k + \lambda_{opt}\, \overline{s}^k$

$\underline{r}^{k+1} := \underline{r}^k - \lambda_{opt}\, \underline{a}^k$

$\overline{w}^{k+1} := \sum_{\ell=1}^{p} C_\ell^T \underline{r}^{k+1}$

$\underline{q}^{k+1} := W^{-1}\overline{w}^{k+1}, \quad \sigma_{k+1} := \langle \underline{q}^{k+1}, \overline{w}^{k+1} \rangle$

$\underline{p}^{k+1} := \underline{q}^{k+1} + \frac{\sigma_{k+1}}{\sigma_k} \underline{p}^k$

$\overline{s}^{k+1} := \sum_{\ell=1}^{p} C_\ell^T \underline{p}^{k+1}$

Parallel Preconditioned Conjugate Gradient Method

Input: Initial guess $\overline{x}^0$

$\underline{r}^0 := \underline{b} - A\overline{x}^0$

$\overline{w}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{r}^0$

$\underline{p}^0 := W^{-1}\overline{w}^0$

$\overline{s}^0 := \sum_{\ell=1}^{p} C_\ell^T \underline{p}^0$

$\sigma_0 := \langle \overline{w}^0, \underline{p}^0 \rangle$

Iteration: $k = 0, 1, \ldots$ (as long as $k < n$, $r^k \neq 0$)

$\underline{a}^k := A\overline{s}^k, \quad \lambda_{opt} := \frac{\sigma_k}{\langle \underline{a}^k, \overline{s}^k \rangle}$

$\overline{x}^{k+1} := \overline{x}^k + \lambda_{opt}\, \overline{s}^k$

$\underline{r}^{k+1} := \underline{r}^k - \lambda_{opt}\, \underline{a}^k$

$\overline{w}^{k+1} := \sum_{\ell=1}^{p} C_\ell^T \underline{r}^{k+1}$

$\underline{q}^{k+1} := W^{-1}\overline{w}^{k+1}, \quad \sigma_{k+1} := \langle \underline{q}^{k+1}, \overline{w}^{k+1} \rangle$

$\underline{p}^{k+1} := \underline{q}^{k+1} + \frac{\sigma_{k+1}}{\sigma_k} \underline{p}^k$

$\overline{s}^{k+1} := \sum_{\ell=1}^{p} C_\ell^T \underline{p}^{k+1}$

Parallel Preconditioned Conjugate Gradient Method

A and W^{-1} are given as type II 'matrices'.

- storage needed for 7 vectors (plus A and W^{-1})

Parallel Preconditioned Conjugate Gradient Method

A and W^{-1} are given as type II 'matrices'.

- storage needed for 7 vectors (plus A and W^{-1})
- 2 vector accumulations (per iteration)

Parallel Preconditioned Conjugate Gradient Method

A and W^{-1} are given as type II 'matrices'.

- storage needed for 7 vectors (plus A and W^{-1})
- 2 vector accumulations (per iteration)
- 2 allreduce-operations

Parallel Preconditioned Conjugate Gradient Method

A and W^{-1} are given as type II 'matrices'.

- storage needed for 7 vectors (plus A and W^{-1})
- 2 vector accumulations (per iteration)
- 2 allreduce-operations
- 1 'local' application of A and W^{-1}

Parallel Preconditioned Conjugate Gradient Method

A and W^{-1} are given as type II 'matrices'.

- storage needed for 7 vectors (plus A and W^{-1})
- 2 vector accumulations (per iteration)
- 2 allreduce-operations
- 1 'local' application of A and W^{-1}
- 2 inner products and 3 saxpy-operations

Parallel Preconditioned Conjugate Gradient Method

A and W^{-1} are given as type II 'matrices'.

- storage needed for 7 vectors (plus A and W^{-1})
- 2 vector accumulations (per iteration)
- 2 allreduce-operations
- 1 'local' application of A and W^{-1}
- 2 inner products and 3 saxpy-operations

How should we choose W^{-1} ???

Debugging MPI Programs

Practical debugging strategies

- run parallel program on single process,

  tests most of functionality, such as I/O

Debugging MPI Programs

Practical debugging strategies

- run parallel program on single process,
  tests most of functionality, such as I/O
- run parallel program with two processes,

Debugging MPI Programs

Practical debugging strategies

- run parallel program on single process,
  tests most of functionality, such as I/O
- run parallel program with two processes,
  or more, such that all functionality can be exercised

Debugging MPI Programs

Practical debugging strategies

- run parallel program on single process,
  tests most of functionality, such as I/O
- run parallel program with two processes,
  or more, such that all functionality can be exercised
- run with smallest problem size that exercises all functionality

Debugging MPI Programs

Practical debugging strategies

- run parallel program on single process,
  tests most of functionality, such as I/O
- run parallel program with two processes,
  or more, such that all functionality can be exercised
- run with smallest problem size that exercises all functionality
  solving a 4×4-system is the same as 1024×1024

Debugging MPI Programs

Practical debugging strategies

- run parallel program on single process,
  tests most of functionality, such as I/O
- run parallel program with two processes,
  or more, such that all functionality can be exercised
- run with smallest problem size that exercises all functionality
  solving a 4×4-system is the same as 1024×1024
- use 'printf'-debugger

Debugging MPI Programs

Practical debugging strategies

- run parallel program on single process,
  tests most of functionality, such as I/O
- run parallel program with two processes,
  or more, such that all functionality can be exercised
- run with smallest problem size that exercises all functionality
  solving a 4×4-system is the same as 1024×1024
- use 'printf'-debugger
- put fflush(stdout); after every printf

Debugging MPI Programs

Practical debugging strategies

- run parallel program on single process,
  tests most of functionality, such as I/O
- run parallel program with two processes,
  or more, such that all functionality can be exercised
- run with smallest problem size that exercises all functionality
  solving a 4×4-system is the same as 1024×1024
- use 'printf'-debugger
- put `fflush(stdout);` after every `printf`
- for point-to-point communication, print data being sent and received

Debugging MPI Programs

Practical debugging strategies

- run parallel program on single process,

  tests most of functionality, such as I/O
- run parallel program with two processes,

  or more, such that all functionality can be exercised
- run with smallest problem size that exercises all functionality

  solving a 4×4-system is the same as 1024×1024
- use '`printf`'-debugger
- put `fflush(stdout);` after every `printf`
- for point-to-point communication, print data being sent and received
- prefix each message with the process rank, sort by rank!

  messages received from different processes do not necessarily arrive in chronological order

Debugging MPI Programs

Practical debugging strategies

- run parallel program on single process,

  tests most of functionality, such as I/O
- run parallel program with two processes,

  or more, such that all functionality can be exercised
- run with smallest problem size that exercises all functionality

  solving a 4×4-system is the same as 1024×1024
- use 'printf'-debugger
- put `fflush(stdout);` after every `printf`
- for point-to-point communication, print data being sent and received
- prefix each message with the process rank, sort by rank!

  messages received from different processes do not necessarily arrive in chronological order
- make sure that all the data structures have been set up correctly

Most frequent sources of trouble

Sequential programming

1. interface problems (types, storage of pointers to data)

Parallel programming

Most frequent sources of trouble

Sequential programming

1. interface problems (types, storage of pointers to data)
2. pointer and dynamical memory management

Parallel programming

Most frequent sources of trouble

Sequential programming

1. interface problems (types, storage of pointers to data)
2. pointer and dynamical memory management
3. logical and algorithmic bugs

Parallel programming

Most frequent sources of trouble

Sequential programming

1. interface problems (types, storage of pointers to data)
2. pointer and dynamical memory management
3. logical and algorithmic bugs

Parallel programming

1. communication

Most frequent sources of trouble

Sequential programming

1. interface problems (types, storage of pointers to data)
2. pointer and dynamical memory management
3. logical and algorithmic bugs

Parallel programming

1. communication
2. races

Most frequent sources of trouble

Sequential programming

1. interface problems (types, storage of pointers to data)
2. pointer and dynamical memory management
3. logical and algorithmic bugs

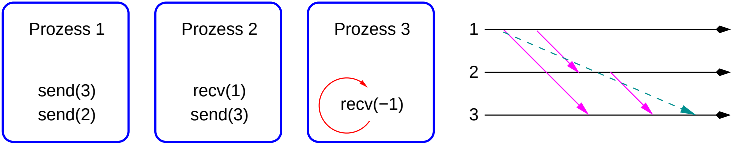
Parallel programming

1. communication
2. races
3. deadlocks

Races

Definition: A race produces an unpredictable program state and behavior due to un-synchronized concurrent executions.
Most often data races occur, which are caused by unordered concurrent accesses of the same memory location from multiple processes.

Example: 'triangle inequality'



Effect: non-deterministic, non-reproducable program running

Communication with MPI

Deadlock I

Time	Process A	Process B
1	`MPI_Send` to B, tag $= 0$	local work
2	`MPI_Send` to B, tag $= 1$	local work
3	local work	`MPI_Recv` from A, tag $= 1$
4	local work	`MPI_Recv` from A, tag $= 0$

Communication with MPI

Deadlock I

Time	Process A	Process B
1	`MPI_Send` to B, tag $= 0$	local work
2	`MPI_Send` to B, tag $= 1$	local work
3	local work	`MPI_Recv` from A, tag $= 1$
4	local work	`MPI_Recv` from A, tag $= 0$

- The program will deadlock, if system provides no buffer.

Communication with MPI

Deadlock I

Time	Process A	Process B
1	`MPI_Send` to B, tag $= 0$	local work
2	`MPI_Send` to B, tag $= 1$	local work
3	local work	`MPI_Recv` from A, tag $= 1$
4	local work	`MPI_Recv` from A, tag $= 0$

- The program will deadlock, if system provides no buffer.
- Process A is not able to send message with tag=0.

Communication with MPI

Deadlock I

Time	Process A	Process B
1	`MPI_Send` to B, tag $= 0$	local work
2	`MPI_Send` to B, tag $= 1$	local work
3	local work	`MPI_Recv` from A, tag $= 1$
4	local work	`MPI_Recv` from A, tag $= 0$

- The program will deadlock, if system provides no buffer.
- Process A is not able to send message with tag=0.
- Process B is not able to receive message with tag=1.

Communication with MPI

Deadlock II

Time	Process A	Process B
1	MPI_Send to B	MPI_Send to A
2	MPI_Recv from B B	MPI_Recv from A

Communication with MPI

Deadlock II

Time	Process A	Process B
1	MPI_Send to B	MPI_Send to A
2	MPI_Recv from B B	MPI_Recv from A

- The program will deadlock, if system provides no buffer.

Communication with MPI

Deadlock II

Time	Process A	Process B
1	MPI_Send to B	MPI_Send to A
2	MPI_Recv from B B	MPI_Recv from A

- The program will deadlock, if system provides no buffer.
- Process A and Process B are not able to send messages.

Communication with MPI

Deadlock II

Time	Process A	Process B
1	`MPI_Send` to B	`MPI_Send` to A
2	`MPI_Recv` from B B	`MPI_Recv` from A

- The program will deadlock, if system provides no buffer.
- Process A and Process B are not able to send messages.
- Order communications in the right way!

Communication with MPI

Example: Exchange of messages

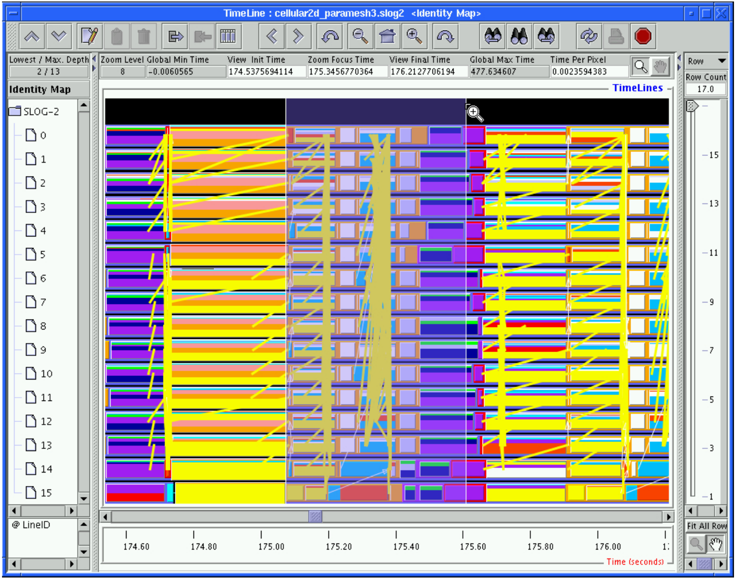
```
if (myrank == 0) {
    MPI_Send( sendbuf, 20, MPI_INT, 1, tag, communicator);
    MPI_Recv( recvbuf, 20, MPI_INT, 1, tag, communicator, &status);
}
else if (myrank == 1) {
    MPI_Recv( recvbuf, 20, MPI_INT, 0, tag, communicator, &status);
    MPI_Send( sendbuf, 20, MPI_INT, 0, tag, communicator);
}
```

- This code succeeds even with no buffer space at all !!!
- **Important note: Code which relies on buffering is considered unsafe !!!**

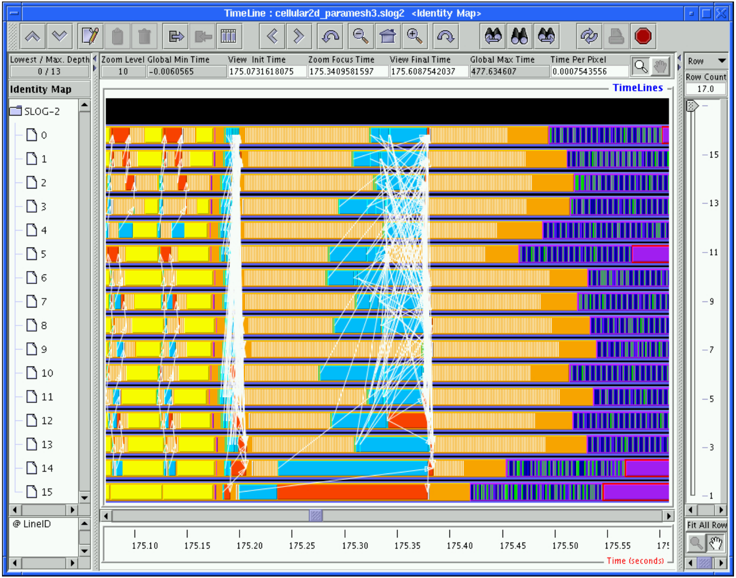
Performance Visualization for Parallel Programs

- MPE is a software package for MPI programmers.
- useful tools for MPI programs, mainly performance visualization
- latest version is called MPE2
- current tools are:
  1. profiling libraries to create logfiles
  2. postmortem visualization of logfiles when program is executed
  3. shared-display parallel X graphics library
  4. . . .

Performance Visualization for Parallel Programs



Performance Visualization for Parallel Programs



Performance Visualization for Parallel Programs