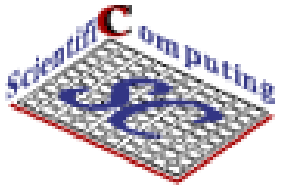
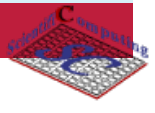


Paralleles Rechnen I

Einführung in das Hochleistungsrechnen – SS 2012
Modellproblem Laplace-Gleichung

Thorsten Grahs, 19. Juni 2012



Beispiele paralleler Anwendungen

- ModellProblem:Laplace-Gleichung
- Direkte Verfahren
- Iterative Verfahren
- Parallelisierungtechniken: Laplace-Gleichung
- Standartbibliotheken (BLAS, LAPACK)

Wärmeleitung

Allgemeines Modell für Wärmeleitung

$$\frac{\partial (c\rho T)}{\partial t}(x,t) + \nabla \cdot \{c(x,t)\,\rho(x,t)\,u(x,t)\,T(x,t) - \lambda(x)\,\nabla T(x,t)\} = f(x,t)$$
$$\forall (x,t) \in \Omega \times \Sigma,$$

mit der Anfangsbedingung

$$T(x,a) = T_0(x), \quad x \in \overline{\Omega},$$

und den Randbedingungen

$$T(x,t) = g(x,t), \qquad (x,t) \in \Gamma_D(t) \times \Sigma, \qquad \Gamma_D(t) \subseteq \partial\Omega,$$
$$\{c\rho u T - \lambda \nabla T\} \cdot \nu = Q(x,t), \qquad (x,t) \in \Gamma_N(t) \times \Sigma, \qquad \Gamma_N(t) = \partial\Omega \setminus \Gamma_D$$

Stationäre Wärmeleitung

Vereinfachungen

- T zeitunabhängig, d.h. $T = T(x)$
- $u = 0$, also kein Stofffluss (Festkörper)

$\Rightarrow$ **stationäre Diffusionsgleichung**

$$
\begin{aligned}
-\nabla \cdot \{\lambda \nabla T(x)\} &= f(x), \quad && x \in \Omega, \\
T(x) &= g(x), && x \in \Gamma_D \subseteq \partial\Omega, \\
-\lambda \nabla T(x)\, \nu &= Q(x), && x \in \Gamma_N = \partial\Omega \setminus \Gamma_D.
\end{aligned}
$$

Mit $\lambda = 1$ ergibt sich die **Poissongleichung**

$$
\begin{aligned}
-\nabla \cdot \nabla T = -\Delta T &= f \quad \text{in } \Omega, \\
T &= g \quad \text{auf } \partial\Omega,
\end{aligned}
$$

Laplace-Gleichung

Laplace Operator

$$\Delta = \sum_{i=1}^{n} \frac{\partial^2}{\partial x_2^2}$$

Vereinfachung: $f = 0$ **(homogene Gleichung)**

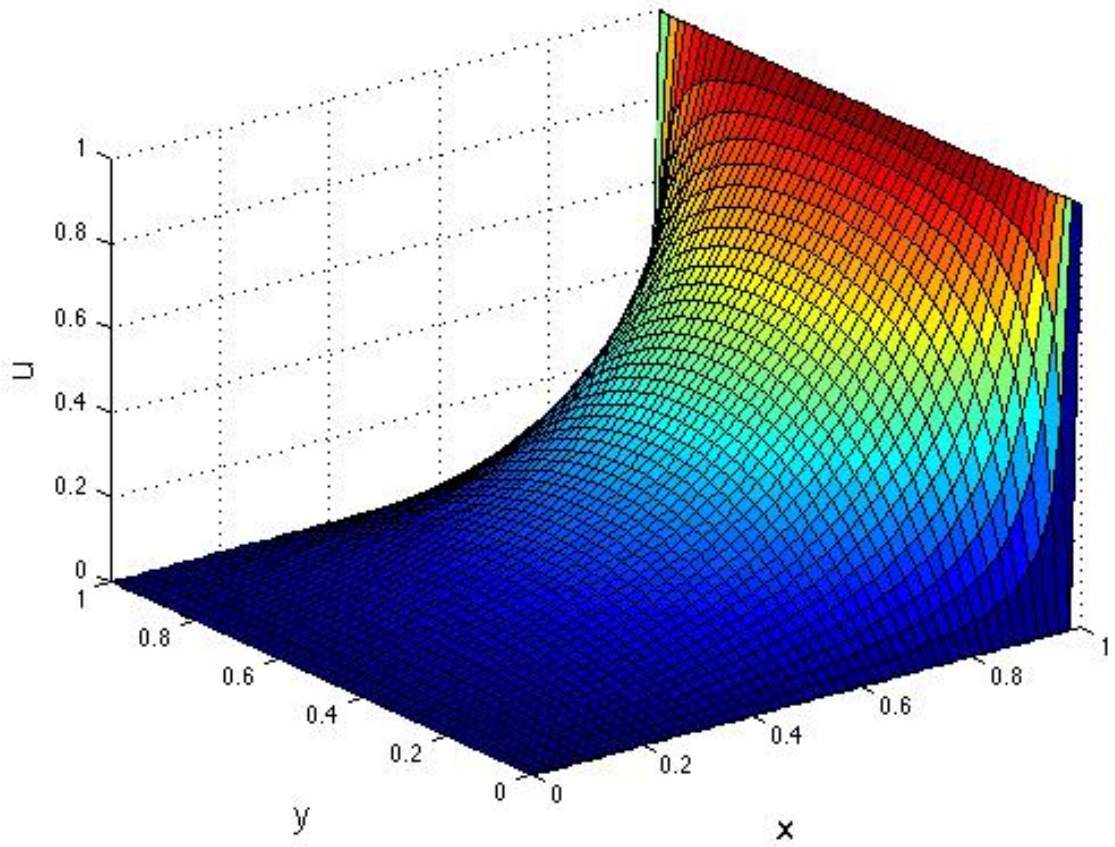
$\Rightarrow$ **Laplace-Gleichung** $\Delta T = 0$

Spezialfall 2D: $n = 2$

$$\Delta T = \frac{\partial^2 T}{\partial x_1^2} + \frac{\partial^2 T}{\partial x_2^2} = 0$$

Lösung der Laplacegleichung

Berechungsgebier 2D



Approximation der Gleichung

Differenzenquotient

$$\partial_x u(x) = \lim_{x \to 0} \frac{u(x+h) - u(x)}{h}$$

Ableitung z.B. aus Taylorentwicklung:

$$u(x+h, y) = u(x, y) + h\partial_x u(x, y) + \frac{h^2}{2}\partial_{x^2}^2 u(x, y) + O(h^3)$$

$$u(x-h, y) = u(x, y) - h\partial_x u(x, y) + \frac{h^2}{2}\partial_{x^2}^2 u(x, y) + O(h^3)$$

Diskretisierung

Approximation 1.Ordnung

- Vorwärtsdifferenz

$$\partial_x u(x,y) \approx \frac{u(x+h,y)-u(x,y)}{h} + O(h)$$

- Rückwärtsdifferenz

$$\partial_x u(x,y) \approx \frac{u(x,y)-u(x-h,y)}{h} + O(h)$$

Diskretisierung

Approximation 2.Ordnung

- zentrale Differenz

$$\begin{aligned}\partial_x u(x,y) \quad \approx \quad & \frac{1}{2h}\left[u(x+h,y) - u(x,y) - \frac{h^2}{2}\partial^2_{x^2}u(x,y)\right] + O(h^2) \\ & -\frac{1}{2h}\left[u(x-h,y) + u(x,y) + \frac{h^2}{2}\partial^2_{x^2}u(x,y)\right] + O(h^2)\end{aligned}$$

$$\partial_x u(x,y) \quad \approx \quad \frac{1}{2h}\left[u(x+h,y) - u(x-h,y)\right] + O(h^2)$$

Diskretisierung d. 2.Ableitung

Approximation 2.Ordnung

$$\partial^2_{x^2} u(x,y) \approx \frac{2}{2h^2}\left[u(x+h,y) - u(x,y) - \partial_x u(x,y)\right] + O(h^2)$$
$$+\frac{1}{2h^2}\left[u(x-h,y) - u(x,y) + \partial_x u(x,y)\right] + O(h^2)$$

$$\partial_x u(x,y) \approx \frac{1}{h^2}\left[u(x+h,y) - 2u(x,y) - u(x-h,y)\right] + O(h^2)$$

Diskretisierung d. Laplace-Operators

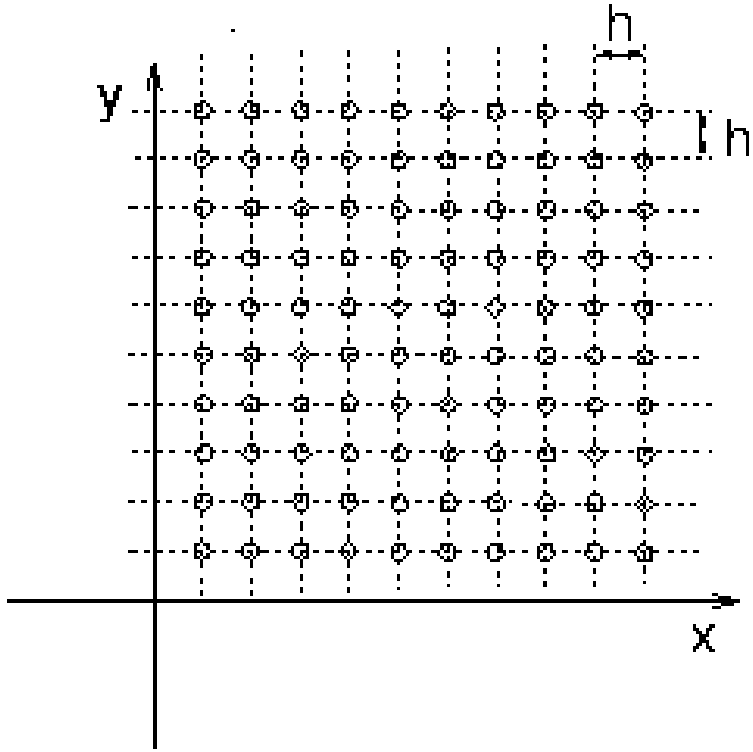
$$\partial^2_{x^2} u(x,y) + \partial^2_{x^2} u(x,y) \approx$$

$$\frac{1}{h^2}\left[u(x+h,y) - 2u(x,y) + u(x-h,y)\right]$$

$$+ \frac{1}{h^2}\left[u(x,y+h) - 2u(x,y) + u(x,y-h)\right]$$

Diskretisierung des Berechnungsgebietes

Berechnungsgebiet zerlegt in ein (nicht notwendigerweise) äquidistanten Gitter (kleine Gitterweite $h = \Delta x = \Delta y$)

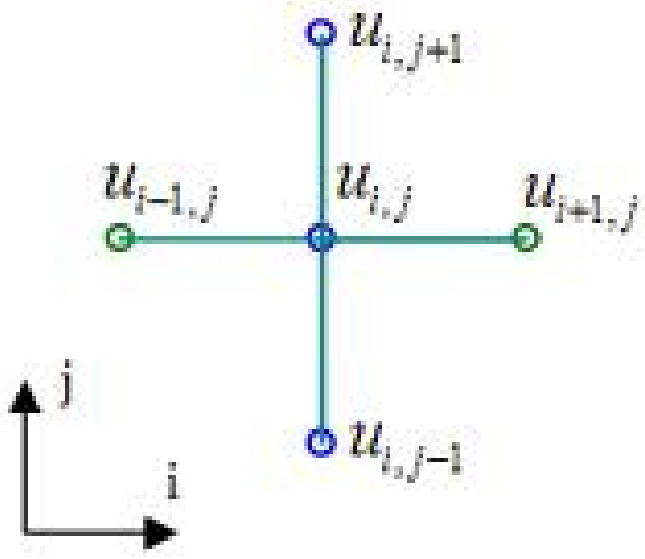


Fünf-Punkte-Stern

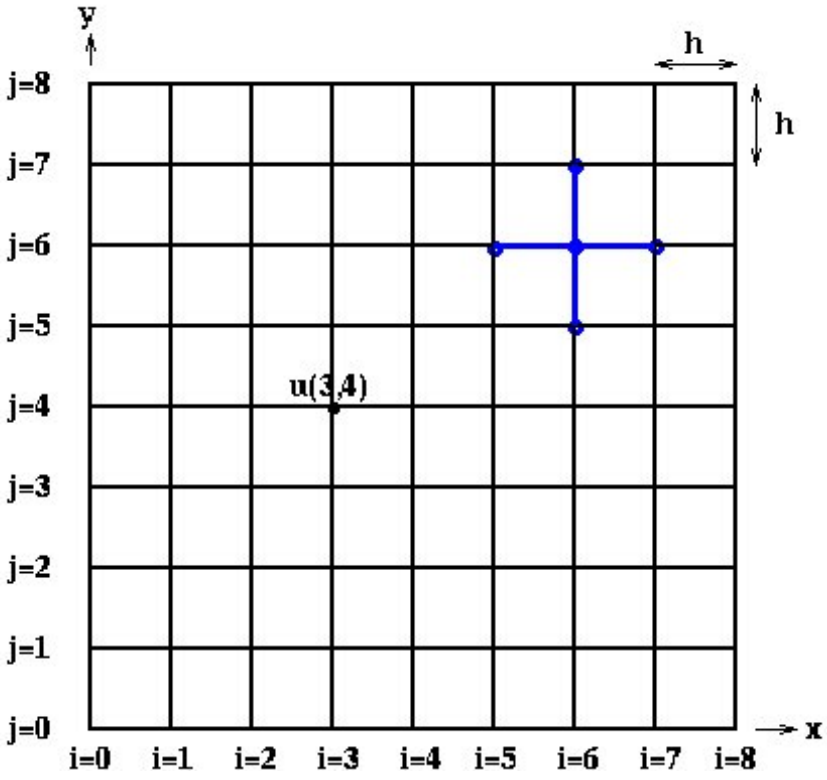
Diskrete Punkte $(x_i, y_j) \quad i = 1, \ldots, n, \quad j = 1, \ldots m$

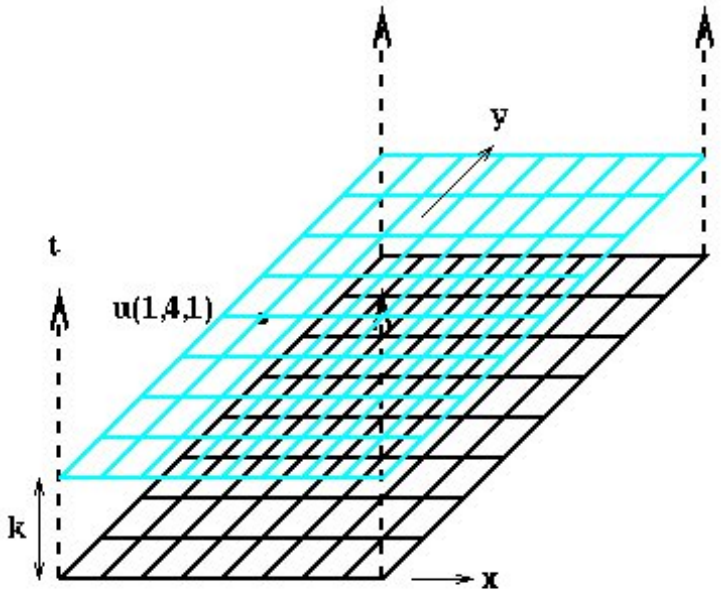
Notation

$u(x_i, y_j) = u_{ij}$



Laplace/Poisson-2D



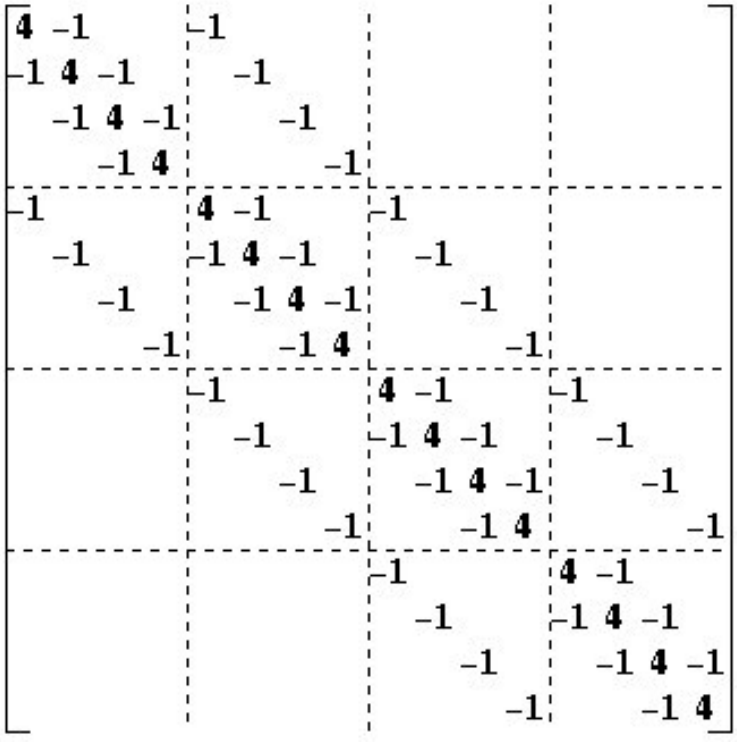


Laplace/Poisson-2D

Berechnung auf Iterationsebene (n+1)

$$u_{i,j}^{n+1} = \frac{1}{4h^2}\left[u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}\right]$$

Resultierende Matrix



Tridiagonal/Bandstruktur

Klassifizierung

Vollbesetzte/Dichte Matrizen

- Voll besetzte Matrizen
  (dense matrices), d.h. wenig/keine Nullen als Einträge
- Direkte Verfahren (Gauss, bzw entsprechende Zerlegungen)
- Bandmatrizen
  Diaqnogalmatrix mit besetzten Nebendiagonalen, ansonsten Nullen
  $\Rightarrow$ Spezielle Algorithmen: Thomas-Algorithmus
- Robuste Standardalgorithmen
  – LAPACK
  – BLAS

Klassifizierung

Dünnbesetzte Matrizen

- Dünnbesetzte Matrizen
  (sparse matrices), d.h. wenige von Null verschiedene Einträge
  spezielles Speicherformat der NichtNull Einträge
- Lösung mit iterativen Verfahren
  – Jacobi
  – Gauss-Seidel
  – Krylov-Unterraum-Methoden

Direkte Verfahren

Gauß-Elimination

Löse Gleichungssystem $\mathbf{A}x = b$

- **Reduktion**

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & \tilde{a}_{22} & \dots & \tilde{a}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ & & & \tilde{\vdots} \\ 0 & 0 & \dots & \tilde{a}_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \tilde{b}_2 \\ \vdots \\ \tilde{\vdots} \\ \tilde{b}_n \end{bmatrix}$$

- **Rückwärtseinsetzen**

$$x_k = \frac{1}{a_{kk}} \left(b_k - \sum_{j=k+1}^{n} a_{kj} x_j \right)$$

Direkte Verfahren

LU/LR-Zerlegung

Löse Gleichungssystem $\mathbf{A}x = b$

- **LU-Zerlegung**

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & \tilde{a}_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & u_{nn} \end{bmatrix}$$

- **Lösen**

$$\mathbf{A}x = b \quad \Rightarrow \mathbf{LU}x = \mathbf{L}(\mathbf{U}x) = \mathbf{L}y = b$$
$$\Rightarrow \mathbf{L}y = b \quad \mathbf{U}x = y$$

Direkte Verfahren

Singulärwert-Zerlegung (Single Value Decomposition(SVD)

Löse Gleichungssystem $\mathbf{A}x = b$

- **Zerlegung** $\mathbf{A} = \mathbf{U}\Lambda\mathbf{V}^\mathbf{T}$ mit

$$U^T U = VV^T = I = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \dots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix} \qquad \Lambda = \begin{bmatrix} \lambda_{11} & 0 & \dots & 0 \\ 0 & \lambda_{22} & \dots & 0 \\ \vdots & \dots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_{nn} \end{bmatrix}$$

- **Lösen**

$$\mathbf{A}x = \mathbf{V}\Lambda^{-1}\mathbf{U}^T b$$

Direkte Verfahren

QR-Zerlegung

Löse Gleichungssystem $\mathbf{A}x = b$

- **Zerlegung** $\mathbf{A} = \mathbf{QR}$ mit
  R obere Dreiecksmatrix
  Q Orthogonale Matrix mit

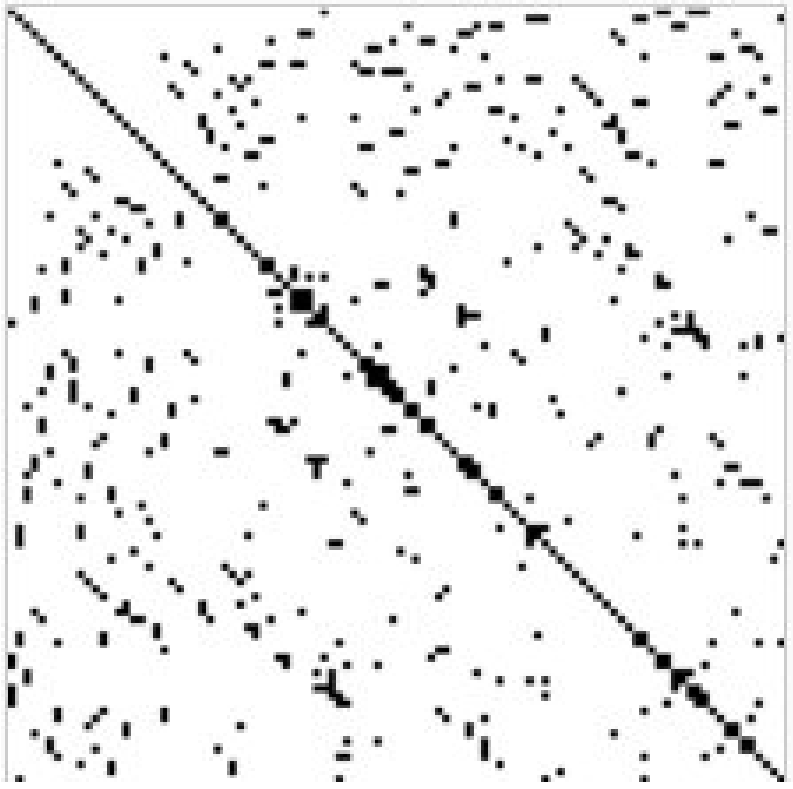
$$Q^T Q = I = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \dots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

- **Lösen**

$$\mathbf{A}x = b \quad \Rightarrow \mathbf{QR}x = b \quad \Rightarrow \mathbf{Q^T QR}x = \mathbf{Q^T}b$$
$$\Rightarrow \mathbf{R}x = \mathbf{Q^T}b$$

Iterative Verfahren

i.Allgemeinen für Dünnbesetzte Matrizen

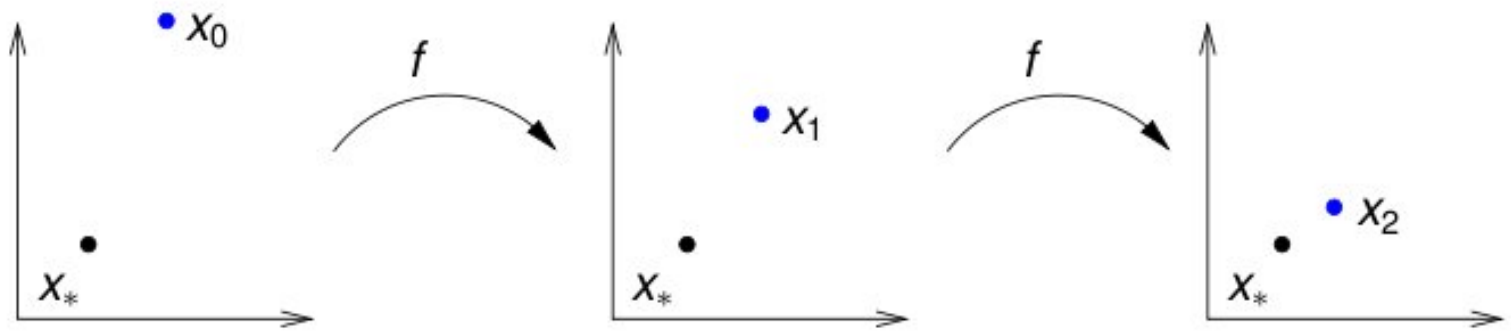


Iterative Verfahren

im Allgemeinen für dünnbesetzte Matrizen

- Braucht meist nur $y = Ax$ oder einfache Zerlegung
- Berechnet sukzessive bessere Approximation x^{k+1}
- Gute Konvergenz ist abhängig von Präkonditionierung
- Die besten Präkonditionierer sind schwer zu parallelisieren

Idee der Iteration



- f ist kontraktiv, sofern $\|f(x) - f(y)\| < \|x - y\|$.
  $\Rightarrow$ Es existiert ein eindeutiger Fixpunk $x^* = f(x^*)$
- Für $x^{k+1} = f(x^k), \quad x^k \to x^*$
- Falls $\|f(x) - f(y)\| < \alpha\|x - y\|, \quad \alpha < 1 \quad \forall x, y$
  $\Rightarrow \|x^k - x^*\| < \alpha\|x - x^*\|$

Iterative Verfahren

Speicherformate dünnbesetzter Matrizen

Speicher der von Null verschiedenen Matrixeinträge, sowie deren Index

- **Yale-Format**
  - Datenvektor *data* der von Null verschiednen Matrixelemente
  - Indexvektor der Indizierung für einen Zeilenübertrag
    (Pos. gibt Zeilennr. an, Eintrag ab welchen Element die Zeile beginnt)
  - Indexvektor der Spaltenindize J der Elemente
- **Index-Format** (scipy) $[A[i[k], j[k]] = data[k]$
  - Datenvektor *data* der von Null verschiednen Matrixelemente
  - Indexvektor der Spaltenindizes I
  - Indexvektor der Zeilenindizes J

Iterative Verfahren

Speicherformate: Beispiel

Matrix $\begin{pmatrix} 1 & 2 & 0 & 0 \\ 0 & 3 & 9 & 0 \\ 0 & 1 & 4 & 0 \end{pmatrix}$

- **Yale-Format**
  - $data = [1\,2\,3\,9\,1\,4]$
  - $I_Z = [0\,2\,4\,6]$
  - $J = [0\,1\,1\,2\,1\,2]$
- **Index-Format** (scipy)
  - $data = [1\,2\,3\,9\,1\,4]$
  - $J = [0\,1\,1\,2\,1\,2]$
  - $I = [0\,0\,1\,1\,2\,2]$

Splitting-Methoden

Iterative Verfahren, für die folgenden Zerlegung existiert:

$\mathbf{A} = \mathbf{M} - \mathbf{N}, \qquad \mathbf{A}, \mathbf{M}, \mathbf{N} \in \mathbb{R}^{n \times n}$

$\mathbf{M}^{-1}$ leicht zu berechnen (Diagonalmatrix)

mit der **Iterationsvorschrift**

$$\mathbf{M}x^{k+1} = \mathbf{N}x^k + b \quad \text{bzw.}$$

$$x^{k+1} = \mathbf{C}x^k + d \quad \text{mit}$$

$$\mathbf{C} := \mathbf{M}^{-1}\mathbf{N} \quad d := \mathbf{M}^{-1}b$$

Konvergenz

Iterative Verfahren, für die folgenden Zerlegung existiert:

- Konvergenz, sofern $\rho(\mathbf{M}^{-1}\mathbf{N}) < 1$
- Ideale Präkonditionierung für Konvergenz: $\mathbf{M} = \mathbf{A}$
- Einfachste Präkonditionierung: $\mathbf{M} = \mathbf{I}$
- Realisitische Präkonditionierung: etwas zwischen
  - Jacobi: $\mathbf{M} = \mathbf{D}$
  - Gauß-Seidel: $\mathbf{M} = (\mathbf{D} - \mathbf{L})^{-1}$

Jacobi-Verfahren

oder **Gesamtschritt-Verfahren**

$\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{R}, \qquad \mathbf{D}, \mathbf{L}, \mathbf{R} \in \mathbb{R}^{n \times n}, \quad \mathbf{D}$ Diagonalmatrix

mit der **Iterationsvorschrift**

$$\mathbf{D}x^{k+1} = \mathbf{L} + \mathbf{R}x^k + b \quad \text{bzw.}$$

$$x^{k+1} = \mathbf{C}x^k + d \quad \text{mit} \qquad \mathbf{C} := \mathbf{D}^{-1}(\mathbf{L} + \mathbf{R}) \quad d := \mathbf{D}^{-1}b$$

$$x_i^{k+1} = \frac{1}{a_i i}\left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij}x_j^k\right), \quad i = 1, \ldots, n.$$

Gauß-Seidel-Verfahren

oder **Einzelschritt-Verfahren**

$\mathbf{A} = \mathbf{D} - \mathbf{L} - \mathbf{R}, \qquad \mathbf{D}, \mathbf{L}, \mathbf{R} \in \mathbb{R}^{n \times n}, \quad \mathbf{D}$ Diagonalmatrix

mit der **Iterationsvorschrift**

$$\mathbf{D} - \mathbf{L} x^{k+1} = \mathbf{R} x^k + b \quad \text{bzw.}$$

$$x^{k+1} = \mathbf{C} x^k + d \quad \text{mit} \qquad \mathbf{C} := (\mathbf{D} - \mathbf{L})^{-1}(\mathbf{R}) \quad d := \mathbf{D} - \mathbf{L}^{-1} b$$

$$x_i^{k+1} = \frac{1}{a_i i} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right), \quad i = 1, \ldots, n.$$

SOR-Verfahren (Successive overrelaxation)

oder **relaxiertes Gauß-Seidel-Vefahren**

Einführung eines Relaxationsparameters $\omega \in (0, 2)$

$\mathbf{A} = \frac{1}{\omega}\mathbf{D} - \mathbf{L} - \mathbf{R} - \frac{1-\omega}{\omega}\mathbf{D}, \qquad \mathbf{D}, \mathbf{L}, \mathbf{R} \in \mathbb{R}^{n \times n}, \quad \mathbf{D}$ Diagonalmatrix

mit der **Iterationsvorschrift**

$$\mathbf{D} - \omega\mathbf{L}x^{k+1} = (1-\omega)\mathbf{D}x^k + \omega\mathbf{R}x^k + \omega b \quad \text{bzw.}$$

$$x^{k+1} = \mathbf{C}x^k + d \quad \text{mit}$$

$$\mathbf{C} := \mathbf{D} - \omega\mathbf{L}^{-1}[(1-\omega)\mathbf{D} + \omega\mathbf{R}] \quad d := \mathbf{D} - \mathbf{L}^{-1}b$$

$$x_i^{k+1} = \frac{\omega}{a_i i}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij}x_j^k\right) + (1-\omega)x_i^k,$$

$i = 1, \ldots, n.$

Krylov-Unterraummethoden

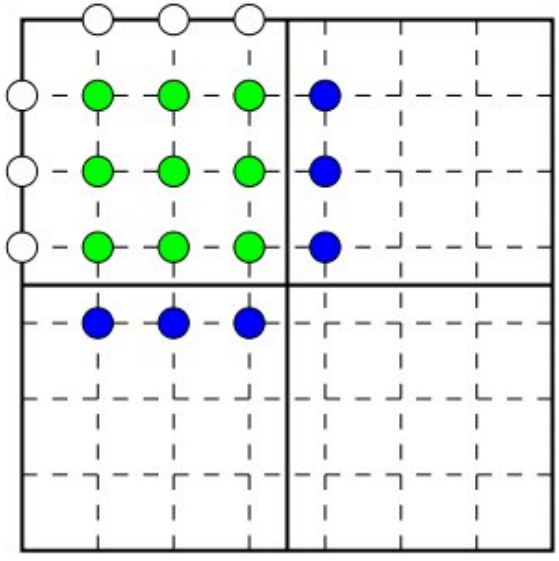
- Matrix-Vektormultiplikation mit **A**
- Spant Krylov-Unterraum

  $K_k(A, b) := [b, \mathbf{A}b, \mathbf{A}^2b, \ldots, \mathbf{A}^{k-1}b]$ auf

- In diesem Unterraum wird die Sukzessive die Lösung x^k gesucht
- Algorithmen:
  – GMRES, CG, BiCG,...

Gebietszerlegung

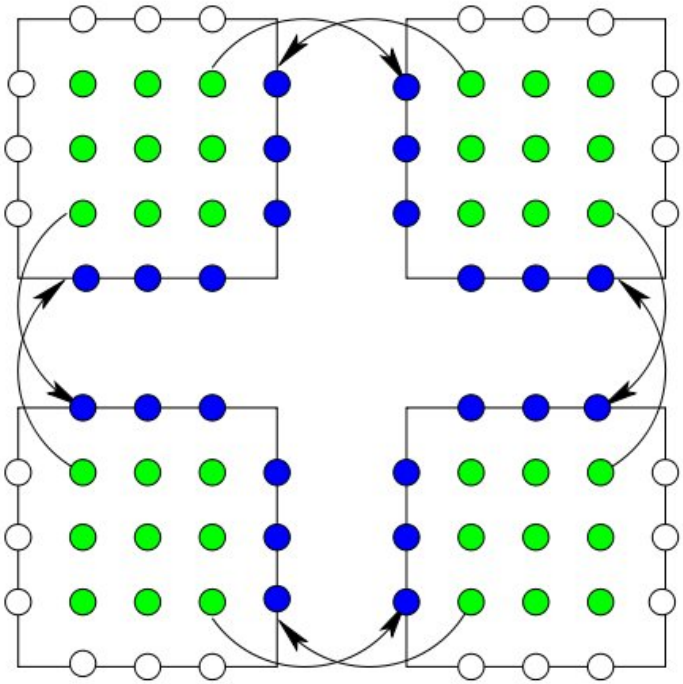
Modell-Problem: Laplace-Gleichung



- ○ **Randwerte**
- ● **Daten auf Prozessor P_0**
- ● **Ghost Cells**

Gebietszerlegung

Modell-Problem: Laplace-Gleichung



- **Kopieren der Ghost cells**

Jacobi-Algorithmus auf 2D Laplace

Modell-Problem: Laplace-Gleichung

```
for step = 1:nsteps
  for i = 2:n-1
    for j = 2:n-1
      u_next(i,j) =
        ( u(i,j+1) + u(i,j-1) + u(i-1,j) + u(i+1,j) )/4*h*h)
    end
  end
  u = u_next;
end
```

Gauß-Seidel-Algorithmus auf 2D Laplace

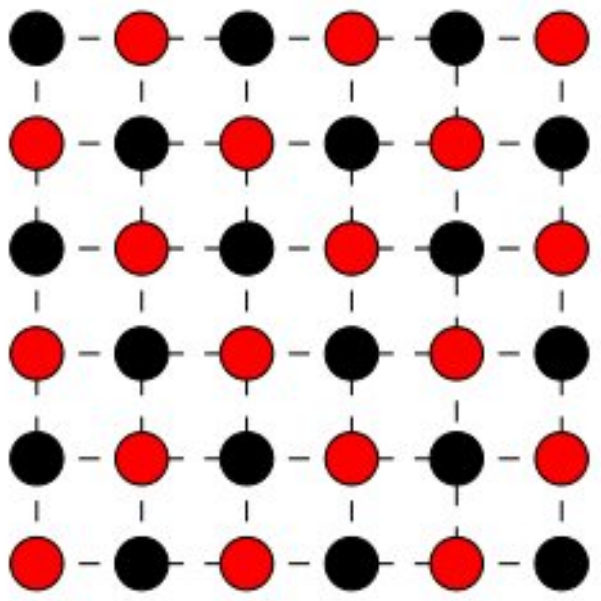
Modell-Problem: Laplace-Gleichung

```
for step = 1:nsteps
  for i = 2:n-1
    for j = 2:n-1
      u(i,j) =
        (u(i,j+1) + u(i,j-1) + u(i-1,j) + u(i+1,j) )/(4*h*h)
    end
  end
end
```

Bei Gauss-Seidel werden Teile der Lösung aus neuer Lösung berechnet. Wie parallelisieren

Red-Black-Gauß-Seidel

Modell-Problem: Laplace-Gleichung



- Rote Werte hängen nur von schwarzen Werten ab
- Schwarze Werte hängen nur von roten Werten ab

Gauß-Seidel-Algorithmus auf 2D Laplace

Modell-Problem: Laplace-Gleichung

```
for i = 2:n-1
  for j = 2:n-1
    if mod(i+j,2) == 0
       u(i,j) = ...
    end
  end
end
for i = 2:n-1
  for j = 2:n-1
    if mod(i+j,2) == 1,
      u(i,j) = ...
    end
  end
```

Paralleler Gauß-Seidel-Algorithmus auf 2D Laplace

Prinzipielles Vorgehen

In jedem Schritt:

- **Kommunikation**
  Sende schwarze Ghost cells
- **Berechnung**
  Update der roten Zellen
- **Kommunikation**
  Sende rote Ghost cells
- **Berechnung**
  Update der schwarzen Zellen

L.A. Pakete – BLAS

Basic Linear Algebra Subprograms (BLAS)

- BLAS-Routinen bilden das Grundgerüst für viel andere LA-Pakete
- BLAS kapselt Basisfunktionalitäten zu Vektor und Matrixoperationen. welche für die jeweiligen Architekturen optimiert werden können
- Routinen bieten die Möglichkeit zur Optimierung der Speicherhierarchie/-nutzung
- In viele Standartdistributionen enthalten, ansonsten über netlib verfügbar.
  `http://www.netlib.org/blas`

Entwicklung BLAS

BLAS 1 (1973-1977) – $O(n^1)$ Op. auf $O(n^1)$ Daten

- Standardbibliothek für 15 grundlegende Vektoroperationen
- Verscheidene Datentypen (S/D/C)
- Beispiel: Routine `DAXPY`
  - Double precision
  - löst $\alpha x + y$
- **Ziele**
  - Grad der Abstraktion zu erhöhen (Programmierung)
  - Portierbares Interface, maschinennahe Implementierung
  - Robuste Algorithmen

Entwicklung BLAS2

BLAS 2 (1984-1986) – $O(n^2)$ Op. auf $O(n^2)$ Daten

- Standardbibliothek für 25 grundlegende Matrix/Vktoroperationen
- Verscheidene Daten- und Matrixtypen
- Beispiel: Routine `DGEMV`
  - Double Precision
  - GEneral Matrix
  - Matrix Vektorprodukt $\mathbf{A}x$
- **Ziele**
  - Optimierung von BLAS1 (ineffizient)
  - Anpasung an neue Vektormaschinen

Entwicklung BLAS3

BLAS 3 (1987-1988) – $O(n^3)$ Op. auf $O(n^2)$ Daten

- Standardbibliothek für 9 grundlegende Matrix/Matrixoperationen
- Verscheidene Daten- und Matrixtypen
- Beispiel: Routine `DGEMM`
  - Double Precision
  - GEneral Matrix
  - Matrix-Matrixprodukt **AB**
- **Ziele**
  - Effiziente Speicher/Cacheausnutzung

Weiterntwicklung BLAS

Verschiedenen Implementationen

- `refblas` – Offizielle Referenzimplementierung von netlib
- `ACML` – AMD Core Math Library,
- `ATLAS` – Automatically Tuned Linear Algebra Software
- `ESSL` – IBMs Engineering and Scientific Subroutine Library
- ...

- `CUBLAS` – Nvidia-Implementierung von BLAS für CUDA
- `XBLAS` – Extra Precise Basic Linear Algebra Subroutines

Warum BLAS?

Betrachte Gauß-Elimination

LU for 2×2:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ c/a & 1 \end{bmatrix} \begin{bmatrix} a & b \\ 0 & d - bc/a \end{bmatrix}$$

Block elimination

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} I & 0 \\ CA^{-1} & I \end{bmatrix} \begin{bmatrix} A & B \\ 0 & D - CA^{-1}B \end{bmatrix}$$

Block LU

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{12} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} = \begin{bmatrix} L_{11}U_{11} & L_{11}U_{12} \\ L_{12}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{bmatrix}$$

Warum BLAS?

Block LU

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{12} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} = \begin{bmatrix} L_{11}U_{11} & L_{11}U_{12} \\ L_{12}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{bmatrix}$$

```
[L11,U11] = small_lu(A);   \\ block LU-Zerlegung von A
 U12 = L11\B;              \\ Lösen Dreiecksmatrix
 L12 = C/U11;              \\      -- " --
 S = D-L21*U12;            \\ Update vom Rang k
[L22,U22] = lu(S);         \\ Finale Faktorisierung
```

- Faktorisierung mit wenig Aufwand und BLAS3

L.A. Pakete – LAPACK

Linear Algebra PACKage (LAPACK)

- Erweiterung von LINPACK (Speziell für Vektorrechner)
- LAPACK umfasst effiziente Routinen zur Lösung
  – linearer Gleichungssysteme
  – linearer Ausgleichsprobleme
  – Eigenwertproblemen.
- Basisfunktionalitäten und -Algorithmendurch BLAS
  (Linear System und Least-Square-Algorithmen basierend nahezu zu 100% auf BLAS)
- Entwicklung seit 1989. Verfügbar unter
  `http://www.netlib.org/lapack`

ScaLAPACK

ScaLAPACK – Scalable Linear Algebra PACKage

- Univ. Tennessee, Univ. Berkeley, Univ. Colorado Denver, NAG Ltd.
- **Jack Dongarra** – EISPACK, LINPACK, BLAS, LAPACK, ScaLAPACK, Netlib, PVM, MPI

- High-Performance Linear Algebra-Routinen
- ausgelegt für parallele Rechner mit veteiltem Speicher (PDMM)
- MPI-Implementierung

LAPACK Funktionalitäten

- General: general (GE), banded (GB), pair (GG), tridiag(GT)
- Symmetric: general (SY), banded (SB), packed (SP),tridiag (ST)
- Hermitian: general (HE), banded (HB), packed (HP)
- Positive definite (PO), packed (PP), tridiagonal (PT)
- Orthogonal (OR), orthogonal packed (OP)
- Unitary (UN), unitary packed (UP)
- Hessenberg (HS), Hessenberg pair (HG)
- Triangular (TR), packed (TP), banded (TB), pair (TG), Bidiagonal (BD)