Parallele Algorithmen und Software für iterative Methoden - Dr.Ing. Martin Bücker - Sommersemester 2003

geTEXt von Christian Terboven
christian@terboven.com

26. Juni 2003

Inhaltsverzeichnis

0 Überblick

Drei Säulen des Erkenntnisgewinns:

- Theorie
- Experiment
- Simulation

Die Bedeutung und Akzeptanz von Simulationen wächst kontinuierlich. Numerische Simulationen beinhalten Methoden, die bei der mathematischen Vorausberechnung von Phänomenen aus Naturwissenschaften und technischen Anwendungen mit Hilfe von modernen Rechnern angewendet werden.

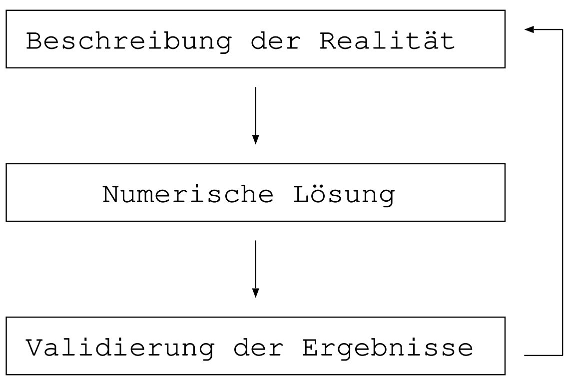
Vorteile:

- Kosten reduzieren
- Umweltbelastung senken
- Experimente präzise vorbereiten
- Rohstoffe sparen
- Versuchapparate schonen
- Untersuchungen dort, wo
  - Experimente undurchführbar sind
  - Experimente zu aufwändig sind
  - Zeitabläufe zu langsam / schnell sind

Nachteile:

- Nur in Verbindung mit Experiment aussagekräftig

0.1 Vorgehensweise der Simulation



Die Güte der Lösung ist abhängig vom mathematischen Modell und Auswahl der Lösungsverfahren.

0.2 Beispiel: Berechnung von Π

$$\Pi = \frac{2 * \Pi * r}{2 * r} = \frac{\text{Kreisumfang}}{\text{Kreisdurchmesser}}$$

Glanzleistung der anthiken Mathematik, Archimedes (287 - 212 v. Chr.): Gerade Strekken (Durchmesser) waren messbar, aber gekrümmte Linien (Umfang) waren damals nicht messbar. Die Rückführung vom "schwierigem" Problem auf ein "leichtes" Problem: wähle Stützpunkte auf dem Kreisumfang und ersetze (krumme) Linie durch (gerade) Sehnen. Die Summe der Sehen entspricht (nähert sich) dem Umfang des Kreises.

0.3 Prinzip der Diskretisierung

Statt unendlich viele Punkte des Kreisbogens betrachte endliche viele Sehnen. Aber: der Vergleich von Archimedes mit der Simulation hinkt! Aschimedes kannte die Funktion und suchte lediglich deren Eigenschaft (Länge). In Simulationen werden Eigenschaften gesucht.

$$\text{Diskretisierung} \rightarrow \text{Gitter}$$

Man erhält nur dann zutreffende Ergebnisse, wenn

- die Anfangsbedingungen hinreichend genau und
- das Gitter fein genug ist.

Falls das Gitter nicht fein genug ist, erhält man schlechte Ergebnisse. Feine Gitter lassen große Datenmengen entstehen.
Fast immer entstehen lineare Gleichungssysteme:

- Finite Differenzen
- Finite Elemente
- Finite Volumen

0.4 Zentrale Aufgabenstellung

Löse $Ax = b$ mit $A \in C^{N \times N}$ und A regulär, d.h. $x^T = A^{-1}b, \quad x, b \in C^N$, außerdem soll

- N groß sein
- das Matrix-Vektor-Produkt $A \cdot y$ für $y \in C^N$ "billig" sein, z.B. in $O(N \cdot log(N))$, nicht $O(N^2)$, z.B.:
  - A dünn besetzt
  - A dicht besetzt, aber mit Struktur

0.5 Einwände

1. Warum Hochschule? Größe der Probleme: $N > 10^9$. Verfahren für kleine N sind nicht notwendig gut geeignet für große N. Standard-Verfahren: Gauß, ... operieren explizit auf der Matrix und benötigen daher $\Omega(N^2)$ Speicher.

2. Warum in Informatik? Parallelverarbeitung (auf p Prozessoren)

   - schafft (diskrete) Informatikprobleme, die im seriellen Fall nicht auftreten
   - "Verfahren für kleine p sind nicht notwendig gut geeignet für große p." Parallelisierung vorhandener serieller Algorithmen ist oft unzureichend für hohe Prozessoranzahlen $\rightarrow$ Entwurf von neuen Algorithmen speziell für Parallelrechner.
   - Modellierung von Algorithmen für Parallelrechner hilft beim Entwurf neuer Algorithmen (allerdings hat man keine reelle Chance neue Algorithmen zu entwerfen, wenn man die Mathematik nicht beherrscht!)

0.6 Speedup und Effizienz

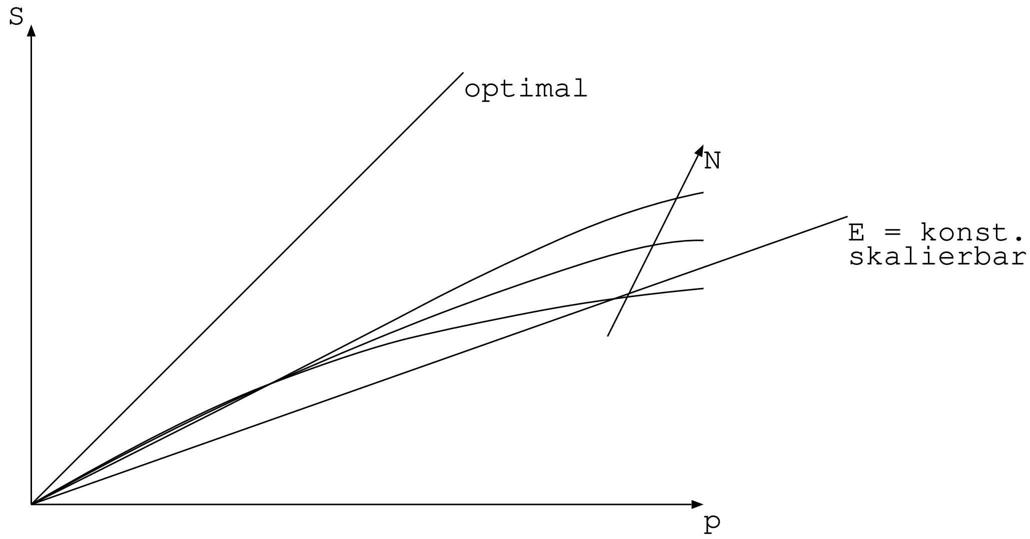
$T_{seq}(N)$ sei die Zeit des schnellsten bekannten Algorithmus zur Lösung eines Problems der Größe N.

$T_{par}(N)$ sei die Zeit zur Lösung desselben Problems auf Parallelrechnern mit p Prozessoren.

$S := \frac{T_{sqe}}{T_{par}}$ heißt Speedup.

$E := \frac{S}{p}$ heißt Effizienz.

Optimal: $S = p \Leftrightarrow E = 1$.



1 "Ungeeignete" Verfahren

Löse $Ax = b$ mit $A \in C^{N \times N}$, regulär, $b \in C^N$.
Gesucht: $x \in C^N$.

1.1 Klassen von Techniken

1. "Direkte" Verfahren: Lösung durch sukzessive Elimination von Variablen:
   - a-priori festgelegte Anzahl von Schritten
   - A wird überschrieben (in-place Algorithmen)
2. "Iterative" (oder "Indirekte") Verfahren: Lösung als Grenzwert einer Folge von Approximationen:
   - unbestimmte Anzahl von Schritten
   - A bleibt unverändert

   Unterklassen:
   - klassische Verfahren (werden heute behandelt)
   - moderne Verfahren (werden im Rest der VL behandelt)

1.2 Definition: dünnbesetzt

Eine Matrix heißt genau dann *dünnbesetzt*, wenn spezielle Techniken angewendet werden können, um aus einer großen Anzahl von Nullen und / oder deren Position Vorteile zu ziehen.
Basisidee: Die Nullen werden nicht gespeichert. Vorstellung einer Datenstruktur in der Übung vom 02.05.2003.

1.3 Direkte Methoden

Ein typischer Vertreter der direkten Methoden ist die Gauß-Elimination:

- Schritt 1: Bestimme die "Zerlegung" der Matrix $A = P \cdot L \cdot U$, wobei P eine $N \times N$-Permutationsmatrix (d.h. $P^{-1} = P^T$) ist, $L = \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{pmatrix}$ eine untere Dreiecksmatrix ($l_{i,i} = 1$) und $U = \begin{pmatrix} * & & * \\ & \ddots & \\ 0 & & * \end{pmatrix}$ eine obere Dreiecksmatrix ist.
- Schritt 2: Löse nun zwei "einfache" Systeme $Ax = b \Leftrightarrow P * L * U = b$ (mit $y := L * U$).

Der Aufwand des Verfährens lässt sich abschätzen zu:

- Operationen: $\frac{2}{3}N^3 + O(N^2)$
- Speicher: N^2

1.3.1 Problemchen:

1. Alle Hauptunterdeterminanten müssen regulär sein.
2. Numerische Stabilität $\rightarrow$ Pivotisierung.
3. Performance kann schlecht sein.

1.3.2 Gravierendes Problem

Falls A dünnbesetzt ist: am Beispiel der Cholesky-Zerlegung für h.p.d. (hermitisch positiv definit) Matrix, d.h. $A = A^H$ und $x^H A x > 0$ für alle $x \neq 0$.

- Schritt 1: Bestimme Zerlegung $A = L^H * L$, wobei $L = \begin{pmatrix} * & & 0 \\ & \ddots & \\ & & * \end{pmatrix}$ eine untere Dreiecksmatrix mit $l_{i,i} > 0$ ist.
- Schritt 2: Löse $Ly = b, \quad L^H x = y$.

Aufwand:

- Operationen: $\frac{1}{3}N^3 + O(N^2)$
- Speicher: $\frac{1}{2}N^2$ falls "in-place".

1.3.3 Resultat

Dünnbesetzte Cholesky-Zerlegung besteht aus 2 Phasen:

1. Symbolische Faktorisierung: bestimme aus Struktur der Matrix (*nicht* aus Werten) eine Permutationsmatrix P, die zu möglichst wenig fill-in in L führt.
2. Numerische Faktorisierung: Mit P aus (1) berechne Zerlegung $B^T A P = LL^H$ und löse zwei Dreicks-Systeme.

Bemerkung: Phase (1) ist ein diskretes Problem (nicht-numerisch) der Informatik. Wie findet man ein geeignetes P? Wie aufwändig ist die Bestimmung des optimalen P? $\rightarrow$ Vorlesung HPC.

Vorteile direkter Methoden:

- Robustheit
- Lange Tradition in kommerzieller Software

Nachteile direkter Methoden:

- Fast keine vernünftige Public-Domain Software
- Oft nicht anwendbar wegen fill-in

1.4 Iterative Methoden

Idee: Spezielle Eigenschaften von A (Dünnbesetztheit, Dichtbesetztheit mit Struktur) können für Operationen der Form $y = A \cdot z$ mit $z \in C$ ausgenutzt werden. $\rightsquigarrow$ Statt $O(N^2)$ Operationen nur $O(N)$ oder $O(N \cdot log(N))$. Löse $Ax = b$, indem A nur in Form dieser Matrix-Vektor-Produkte verwendet wird.

1.4.1 Allgemeine Vorgehensweise klassischer iterativer Verfahren

Im n-ten Schritt:

- $x_n :=$ Approximation an exakte Lösung. $x_* := A^{-1}b$. $\quad r_n := b - Ax_n$ heißt Residuum.
- Ziel: $r_n \to 0$ für $n \to \infty$ (möglichst schnell, $n << N$), weil dann $b - Ax = 0$ und dort ist $x_n = x_*$.
- Bestimme x_n durch geeignete Wahl von r_n.

1.4.2 Allgemeiner Ansatz klassischer iterativer Verfahren

$M x_{n+1} = S x_n + b$, wobei $A = M - S$, “splitting” von A.
Wahl von M so, dass Systeme mit M “einfach” gelöst werden können.

Beispiel: $A = D + L * U$

- Jacobi: $M_{jac} = D$, $S_{jac} = -(L + U)$
- Gauß-Seidel: $M_{gs} = D + L$, $S_{gs} = -U$

1.4.3 Konvergenzverhalten

Iteration $x_{n+1} = M^{-s} \cdot S \cdot X_n + M^{-1} \cdot b$ soll schnell konvergieren. Betrachte dazu den Fehler $e_n := x_n - x_*$.

$$\begin{aligned}
x_{n+1} &= M^{-1} \cdot S \cdot (e_n + x_*) + M^{-1} \cdot b \\
&= M^{-1} \cdot S \cdot x_* + M^{-1} \cdot b + M^{-1} \cdot S \cdot e_n \\
&\rightsquigarrow e_{n+1} = x_{n+1} - x_* = M^{-1} \cdot S \cdot e_n = B \cdot e_n \\
&= B^n \cdot e_1.
\end{aligned}$$

Was passiert bei wiederholter Anwendung einer Matrix B auf Vektor v?

1. Fall 1: v ist Eigenvektor, d.h. $B \cdot v = \lambda \cdot v$. $B^n \cdot r = \lambda^n \cdot v$. Das strebt für $n \to \infty$ gegen 0, wenn $|\lambda| < 1$, bei $|\lambda| > 1$ ist es divergent.
2. Fall 2: v ist kein Eigenvektor. Annahme:

$$\begin{aligned}
v &= \sum_{i=1}^{k} \alpha i \cdot B^n \cdot v_i = sum_{i=1}^{k} \alpha_i \cdot \lambda_i^n \cdot v_i \\
B^n v &= \sum_{i=1}^{k} \alpha_i \cdot B^n \cdot v_i = \sum_{i=1}^{k} \alpha_i \cdot \lambda_i^n \cdot v_i
\end{aligned}$$

Das strebt für $n \to \infty$ gegen 0, wenn $|\lambda_i| < 1$ $(i = 1, \ldots k)$, bei $|\lambda| > 1$ ist es divergent.

Daher ist man am sog. *Spektralradius* einer Matrix B interessiert. $\varphi(B) := max|\lambda_i|$, λ_i ist Eigenwert von B.

Zurück zum Konvergenzverhalten: $e_{n+1} = (M^{-1} \cdot S)^n \cdot e_1$ ist konvergent, falls $\varphi(M^{-1} \cdot S) < 1$.

- Schnelle Konvergenz, falls $\varphi(M^{-1} \cdot S) << 1$.
- Schnelle Konvergenz, falls $\varphi(M^{-1} \cdot S) \approx 1$.

2 Grundlagen moderner iterativer Verfahren

Löse $Ax = b$ iterativ mit Approximationen $x_0, ..., x_n \in C^N$.
Ziel: Residuum $r_n := b - Ax_n\ to 0 (n \to \infty)$ möglichst schnell.

Wie messe ich, wie weit r_n von 0 entfernt ist? Eine (Vektor-)Norm ist eine Funktion $|| \cdot || : C^N \to R$, die einem Vektor eine "Länge" zuordnet. Eigenschaften von Normen: für alle $x, y \in C^N$ und $\alpha \in C$ gilt:

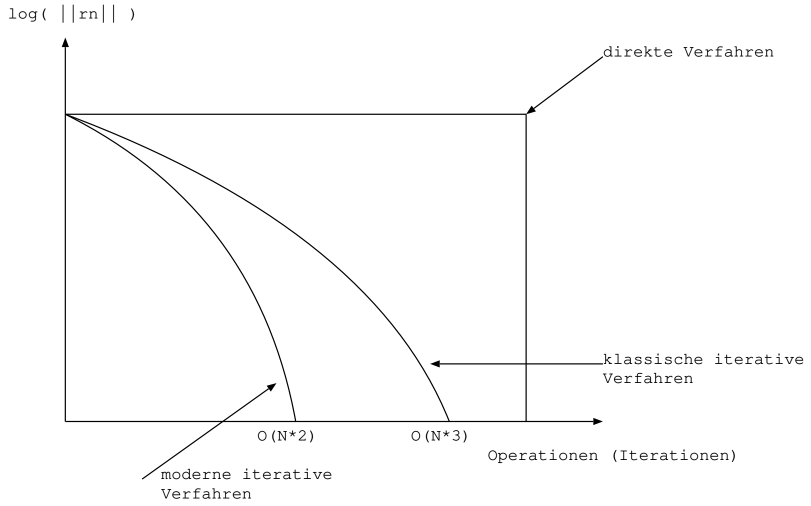
- $||x|| \geq 0$ und $||x|| = 0$ nur für $x = 0$
- $||x + y|| \leq ||x|| + ||y||$ (Dreiecksungleichung)
- $||\alpha * x|| = |\alpha * ||x||$.

Eine wichtige Klasse von Normen sind die Hölder- oder p-Normen:

$$||x||_p := (\sum_{i=1}^{k} |x_i|^p)^{\frac{1}{p}} \quad (1 \leq p \leq \infty)$$

wobei $||x||_\infty := max(|x_i|), (1 \leq i \leq n)$ (betragsgrößte Komponente).
Soweit nicht anders erwähnt, wird hier immer die 2-Norm (Euklidische Norm) verwendet.

Also für uns interessant: Beobachtung von $||r_n||_2$.



$\rightsquigarrow$ N wächst rasant und iterative Verfahren werden immer wichtiger.

2.1 Background Lineare Algebra

$$A^H := \overline{A^T} = \overline{A}^T$$

Analog zu

$$\begin{aligned} (A \cdot B)^{-1} &= B^{-1} \cdot A^{-1} \quad \text{für quadratische Matrizen} \\ (A \cdot B)^H &= B^H \cdot A^H \quad \text{für rechteckige Matrixzen mit passenden Dimensionen} \end{aligned}$$

Schreibweise: $A^{-H} := (A^{-1})^H = (A^H)^{-1}$

$$x^H y = \sum_{i=1}^{n} \overline{x_i} \cdot y_i = \overline{y^H \cdot y} \quad \text{inneres Produkt von } x, y \in C^N\text{, damit ist } ||x||_2 = \sqrt{x^H x}$$

Orthogonalität:

1. Vektoren x, y orthogonal: $\Leftrightarrow x^H y = 0$
2. Mengen von Vektoren X und Y orthogonal: $\Leftrightarrow x^H y = 0 \quad \forall x \in X, \forall y \in Y$
3. Eine Menge M von Vektoren ist orthogonal: $\Leftrightarrow x, y \in M, x \neq y \quad \Rightarrow x^H y = 0$. Sie ist orthonormal, wenn zusätzlich $||x|| = 0 \quad \forall x \in M$.

Eine quadratische Matrix Q heißt unitär, $\Leftrightarrow Q^H = Q^{-1}$ bzw. $Q^H \cdot Q = I$.
Innere Produkte bleiben unter Multiplikation unitärer Matrizen invariant:

$$x^H y = x^H \cdot Q^H \cdot Q \cdot y = (Qx)^H \cdot (Qy)$$

Sei $A = [a_1\, a_2\, \ldots\, a_N] \in C^{N\times N}$ mit Spalten $a_i \in C$. Für $z \in C^N$ gilt:

$$A \cdot z = \sum_{i=1}^{n} z_i \cdot a_i \quad \text{wobei } z = \begin{pmatrix} z_1 \\ \vdots \\ z_n \end{pmatrix}$$

Also: nicht A wirkt auf z, sondern eher umgekehrt.
Spaltenraum von A: range(A) = span($a_1,\ldots,a_n$) = $\{y \in C^N | y = A\,z, \quad z \in C\}$.
$\rightsquigarrow$ Bei Lösung von $Ax = b$ interpretiere $x = A^{-1}\cdot b$ nicht als A^{-1} angewendet auf b, sondern $x = A^{-1}\cdot b$ ist Koeffizientenvektor in der Darstellung von b bzgl. der durch die Spalten von A gebildeten Basis.

2.2 Grundlagen

Bei klassischen Methoden werden Approximationen x_n komponentenweise "verbessert". Sei $e_i = (0,..,0,1,0,..,0)^T \in R^n$ mit 1 an Position i. Dann wurden die i-ten Komponenten sukzessive von der folgenden Form aufaddiert:

$$x_n = x_{n-1} + \alpha_i \cdot e_i \quad \text{so dass } r_n^H \cdot e_i = 0$$

d.h. Update i-ter Komponente so, dass i-te Komponente von r_n verschwindet.

x_i aus Bedingung:

$$\begin{aligned}
r_n^H \cdot e_i = 0 \quad &\Leftrightarrow \quad (b - A\,x_n)^H \cdot e_i = 0 \\
&\Leftrightarrow \quad (b - A\,x_{n-1} - \alpha_i \cdot A \cdot e_i)^H \cdot e_i = 0 \\
&\Leftrightarrow \quad (r_{n-1} - \alpha_i \cdot A \cdot e_i)^H \cdot e_i = 0 \\
&\Leftrightarrow \quad \underline{r_{n-1}^H \cdot e_i} = \underline{(\alpha_i \cdot A \cdot e_i)^H \cdot e_i} = e_i^H \cdot \alpha_i \cdot A \cdot e_i \\
&\Leftrightarrow \quad \boxed{x_i = (e_i^H \cdot A \cdot e_i)^{-1} \cdot e_i^H \cdot r_{n-1}}
\end{aligned}$$

Kann man nicht mehr als eine Komponente gleichzeit einbeziehen?
$\rightsquigarrow$ Räume statt e_i.
$\mathcal{K} = span(e_i)$ und $\mathcal{L} = span(e_i)$
$x_n = x_{n-1} + \mathcal{K}$ so, dass $r_n = b - A\,x_n \perp \mathcal{L}$
$\mathcal{K}$ ist Suchraum, in dem die Approximationen konstruiert werden (search subspace).
$\mathcal{L}$ ist Teilraum von Bedingungen (subspace of constraints).

Sei dim($\mathcal{K}$)= m und dim($\mathcal{L}$)= m.
$\mathcal{K}$=span($v_1,\ldots,v_m$) und $\mathcal{L}$=span($w_1,\ldots,w_n$)).
(N × m)-Matrizen $V_m := [v_1 \ldots v_m]$ und $W_m := [w_1 \ldots w_m]$.
Dann ist $x_n = x_{n-1} + V_m\,y_m$, wobei $y_m \in C^m$ über die Bedingung für $\mathcal{L}$ bestimmt ist:

$$\begin{aligned}
r_n \perp \mathcal{L} \quad &\Leftrightarrow \quad w^H\,r_n = 0 \quad \forall w \in \mathcal{L} \\
&\Leftrightarrow \quad W_m^H\,r_n = 0 \\
&\Leftrightarrow \quad W_m^H \cdot (b - A\,x_n) = 0 \\
&\Leftrightarrow \quad W_m^H(\underbrace{b - A\,x_{n-1}}_{r_{n-1}} - A \cdot V_m \cdot y_m) = 0
\end{aligned}$$

$$\begin{aligned}
&\Leftrightarrow \quad W_m^H \cdot r_{n-1} = \underline{W_m^H \cdot A \cdot W_m \cdot y_m} \\
&\Leftrightarrow \quad \boxed{y_m = (W_m^H \cdot A \cdot V_m)^{-1} \cdot W_m^H \cdot r_{n-1}} \\
&\rightarrow \quad \text{Projektionsverfahren.}
\end{aligned}$$

Falls $\mathcal{K} = \mathcal{L}$, heißt dies eine orthogonale Projektion.
Falls $\mathcal{K} \neq \mathcal{L}$, heißt dies eine oblique Projektion.
x_n existiert nur, falls $W_m^H \cdot A \cdot V_m$ nichtsingulär ist. Dies ist nicht immer der Fall, selbst wenn A nichtsingulär ist.

2.3 Lemma

Die Matrix $W_m^H \cdot A \cdot V_m$ ist nichtsingulär für jede Basis V_m und Basis W_m von $\mathcal{K}$ und $\mathcal{L}$, falls entweder

1. A ist positiv definit und $\mathcal{K} = \mathcal{L}$, oder
2. A ist nichtsingulär und $\mathcal{L} = A \cdot \mathcal{L}$.

In diesem allgemeinen Rahmen der Projektionsmethoden gilt das folgende Optimalitätsergebnis:

2.4 Satz

Ein Vektor x_n ist die auf x_{n-1} folgende Approximation einer (obliquen) Projektionsmethode auf $\mathcal{K}$ entlang $\mathcal{L} = A \cdot \mathcal{K}$ g.d.w:

$$||b - A \cdot x_n||_2 = \min_{x \in x_{n-1} + \mathcal{K}} ||b - Ax||_2$$

Der Vektor x_n minimiert die Norm des Residuum über alle Vektoren aus $x_{n-1} + \mathcal{K}$.

3 Konstruktion einer Basis für Suchraum K (Teil 1: Arnoldi)

Sei $dim(\mathcal{K}) = m$

Dann ist $x_n = x_{n-1} + V_m y_m$, wobei $y_m \in C^m$ über die Bedingung für $\mathcal{L}$ bestimmt ist:

$$\begin{aligned}
r_n \perp \mathcal{L} \quad &\Leftrightarrow \quad W^H r_n = 0 \quad \forall w \in \mathcal{L} \\
&\Leftrightarrow \quad W_m^H (b - Ax_n) = 0 \\
&\Leftrightarrow \quad W_m^H (\underbrace{b - Ax_{n-1}}_{r_{n-1}} - AV_m y_m) = 0 \\
&\Leftrightarrow \quad W_m^H r_{n-1} = \underline{W_m^H A V_m} y_m \\
&\Leftrightarrow \quad \boxed{y_m = (W_m^H A V_m)^{-1} W_m^H r_{n-1}} \\
&\Rightarrow \quad \text{Projektionsverfahren.}
\end{aligned}$$

4 Konstruktion einer Basis für Suchraum K (Teil 2: Lanczos)

5 Konstruktion einer Basis für Suchraum K (Teil 2: Lanczos)

6 Parallele Programmiermodelle

6.1 Auswahl eines parallele Programmiermodells

Sich widersprechende Ziele:

- Programmierung soll "leicht" sein
- Programme sollen mit "wenig" Aufwand über verschiedene Plattformen portablen sein
- Programmierer soll Kontrolle über Performance haben

Mögliche Unterteilungen:

- Daten ⇔ Aufgabeparallelismus: Parallele Ausführung der gleichen Aufgabe auf verschiedenen Daten ⇔ parallele Ausführung verschiedener Aufgaben auf gleichen / verschiedenen Daten
- Explizite / implizite Parallelisierung: Programmierer spezifiziert explizit die Aktionen zur parallelen Ausführung ⇔ Programmierer spezifiziert abstrakte Spezifikation, die Aktionen zur parallelen Ausführung implizieren
- Gemeinsamer ⇔ verteilter Speicher: Austausch von Daten zwischen verschiedenen Prozessoren über einzelnen gemeinsamen Speicher ⇔ durch Interaktionen von / zu unterschiedlichen lokalen Speichern

  Tabelle...

  Am Beispiel von Basisoperationen vom Krylov-Teiler-Verfahren. Betrachte für gegebene Matrix $A \in R^{NxN}$ und Vektoren $u, v, w \in R^N$ die Sequenz von Anweisungen

  - $u = A * v$
  - $\alpha = w^T * v$
  - $w = \alpha * u + v$

  Die Implementierung von Message Passing und Multithreading kann mit Hilfe von Bibliotheken und/oder Compiler-/Laufzeitsystemen erfolgen.

  Tabelle...

6.2 Message Passing und das Modell des verteilten Speichers

Logische Sicht einer Architektur mit verteiltem Speicher:

- Jeder Prozessor besitzt Zugriff nur auf seinen lokalen Speichern.
- Datenaustausch durch explizite Kopieroperationen (Senden / Empfangen) über Verbindungsnetzwerk (Fig. 5.3)

Wichtige Eigentschaften:

- Verteilter Adressraum
  * explizite Partitionierung von Datenstrukturen notwendig (aber auch Kontrolle über Lokalität von Daten)
  * Bei allen Interaktionen zwischen Prozessoren sind mindestens zwei Prozessoren beteiligt (zweiseitige Kommunikation)
- explizite Parallelisierung
  * Das System führt keine Aktion zur Parallelisierung aus, die nicht explizit programmiert wurde.

Übliche Programmierung im Single Programm Multiple Data -Ansatz (SPMD) = Programm, dass von verschiedenen Prozessoren ausgeführt wird, ist identische (parametrisiert durch eindeutige Nummer eines Prozesses).

Der MPI-Standard (www.mpi-forum.org) vereinheitlicht seit Mitte 1990 eine große Anzahl von message-passing Bibliotheken

In MPI können Prozessoren in logische Gruppen eingeteilt werden (Communicator), default: MPI_COMM_WORLD = alle Prozessoren, auf denen das Programm ausgeführt wird.

Vorgehensweise bei MPI-Parallelisierung:

1. Verteilung der Datenstrukturen
2. Index-Transformation: globaler Index $\rightarrow$ Prozess-ID + lokalem Index
3. Verteilung der Berechnung
4. Kommunikation, wenn Daten von anderen Prozessen benötigt werden

6.3 Multithreading uns das Modell des gemeinsamen Speichers

6.4 MPI oder OpenMP?