Parallele Algorithmen und Software für iterative Methoden - Dr.Ing. Martin Bücker[1] - Sommersemester 2003

geTEXt von Christian Terboven
christian@terboven.com

31. Juli 2003

[1]Diese Mitschrift soll bis Ende August komplett sein. Darüber hinaus hat der Dozent bis Anfang WS-2003 ein eigenes Skriptum in Aussicht gestellt.

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.

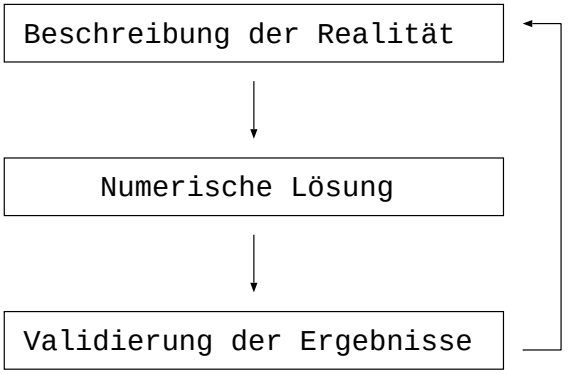
Vorteile:

- 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

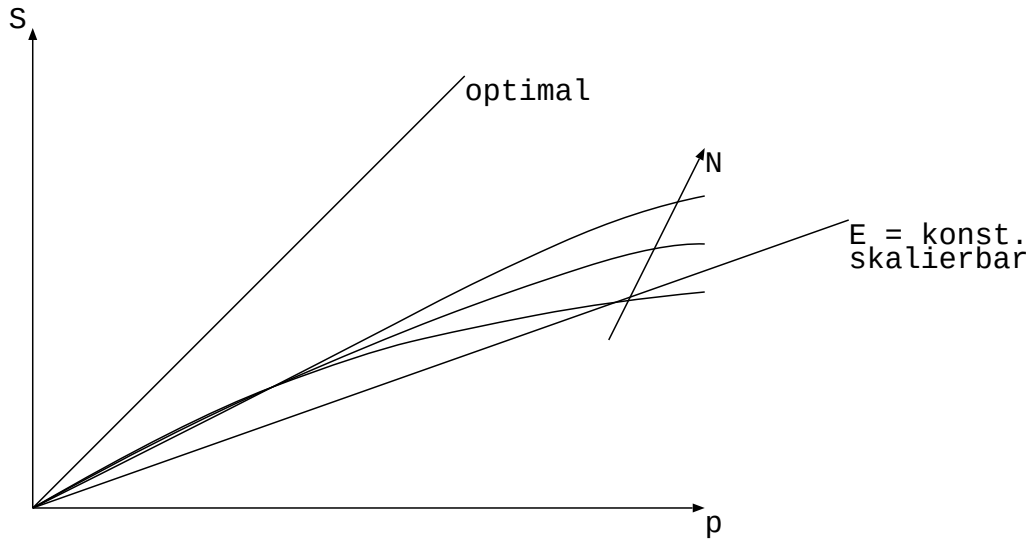
$T_{seq}(N)$ sei die Zeit des schnellsten bekannten Algorithmus zur Lösung eines Problems der Größe N.

$T_{par}(N)$ 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: $S = p \Leftrightarrow E = 1$.



Lecture 1

Inadequate Methods

Table 1.1: Performance (in MFLOPs/s) of three different implementations of Gaussian elimination applied to linear systems of size N on a SUN ULTRA with theoretical peak performance of 334 MFLOPs/s written in FORTRAN

Implementation	$N = 100$	$N = 200$	$N = 300$	$N = 400$	$N = 500$	$N = 600$
naive	31.9	26.0	13.1	7.7	7.4	5.9
loop-arranged	71.3	78.2	57.4	47.8	42.8	39.9
Block (BLAS3)	90.1	108.4	124.2	129.1	139.3	140.5

$[\mathbf{x}_n, \mathbf{r}_n] =$ RELAXATION$(A, \mathbf{b}, \mathbf{x}_0)$ If $A \in \mathbb{C}^{N\times N}$ with splitting $A = M - S$ with nonsingular M, this algorithm computes approximations $\mathbf{x}_n$ (with corresponding residuals $\mathbf{r}_n$) to the solution of the linear system $A\mathbf{x} = \mathbf{b}$ for any starting vector $\mathbf{x}_0$.

1: Choose $\mathbf{x}_0 \in \mathbb{C}^N$, set $\mathbf{r}_0 \leftarrow \mathbf{b} - A\mathbf{x}_0$, and solve $M\mathbf{z}_0 = \mathbf{r}_0$
2: **for** $n = 1, 2, 3, \ldots$ **do** {until convergence}
3: $\quad \mathbf{x}_n \leftarrow \mathbf{z}_{n-1} + \mathbf{x}_{n-1}$
4: $\quad \mathbf{r}_n \leftarrow \mathbf{b} - A\mathbf{x}_n$
5: $\quad$ Solve $M\mathbf{z}_n = \mathbf{r}_n$
6: **end for**

Figure 1.1: Relaxation iteration

Theorem 1.1. *An iterative scheme of the form*

$$M\mathbf{x}_n = S\mathbf{x}_{n-1} + \mathbf{b}$$

for the solution of $A\mathbf{x} = \mathbf{b}$ with nonsingular coefficient matrix $A = M - S$ converges to the exact solution $\mathbf{x}_* = A^{-1}\mathbf{b}$ for any starting vector $\mathbf{x}_0$, if M is nonsingular and

$$\rho(M^{-1}S) < 1.$$

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 $x^H Ax > 0$ 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 $B^T AP = LL^H$ 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

$Mx_{n+1} = Sx_n + b$, 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 $\varphi(M^{-1} \cdot S) \approx 1$.

Lecture 2

Fundamentals of Modern Iterations

$\mathrm{x}_n = \textsc{Projection}(A, \mathrm{b}, \mathrm{x}_0)$ If $A \in \mathbb{C}^{N\times N}$ and x_0 is a starting vector, this algorithm computes approximations x_n to the solution of the linear system $A\mathrm{x} = \mathrm{b}$.

1: Choose $\mathrm{x}_0 \in \mathbb{C}^N$
2: **for** $n = 1, 2, 3, \ldots$ **do** {until convergence}
3: Choose subspaces $\mathcal{K}$ and $\mathcal{L}$
4: Choose basis $V_m = [\mathrm{v}_1\ \mathrm{v}_2\ \cdots\ \mathrm{v}_m]$ of $\mathcal{K}$ and choose basis $W_m = [\mathrm{w}_1\ \mathrm{w}_2\ \cdots\ \mathrm{w}_m]$ of $\mathcal{L}$
5: $\mathrm{r}_{n-1} \leftarrow \mathrm{b} - A\mathrm{x}_{n-1}$
6: $\mathrm{y}_m \leftarrow (W_m^H A V_m)^{-1} W_m^H \mathrm{r}_{n-1}$
7: $\mathrm{x}_n \leftarrow \mathrm{x}_{n-1} + V_m \mathrm{y}_m$
8: **end for**

Figure 2.1: General form of a projection method

Theorem 2.1 (Optimality of projection method with $\mathcal{L} = A\mathcal{K}$). *A vector x_n is the next approximation of a projection method onto the search subspace $\mathcal{K}$ along the subspace of constraints $\mathcal{L} = A\mathcal{K}$ if and only if*

$$\|\mathrm{b} - A\mathrm{x}_n\| = \min_{\mathrm{x}\in \mathrm{x}_{n-1}+\mathcal{K}} \|\mathrm{b} - A\mathrm{x}\|. \tag{2.1}$$

Lecture 2

Fundamentals of Modern Iterations

Classical iterations based on relaxation of coordinates as discussed in the previous lectures are currently used as building blocks in multigrid algorithms. However, they are rarely used as pure iterative methods for the solution of linear systems. By revisiting an example of a classical relaxation method in Sec. 2.1, a motivation for a more general framework of projection methods is given. This unifying framework is used to introduce the general background of modern iterative methods. The overall idea of a projection method is to construct the approximations in a certain subspace. The actual approximation follows from imposing certain restrictions subject to another subspace. Projection methods are described in Sec. 2.2. Without being too specific about the two subspaces, a result is given in Sec. 2.3 that provides insight into the convergence behavior of a particular class of projection methods.

2.1 Jacobi Iteration Revisited

The key idea of the Jacobi method introduced in the previous lecture is to successively zero out the components of the residual vector. In a single step of the Jacobi method, the next approximation $\mathbf{x}_n$ is obtained from the current approximation $\mathbf{x}_{n-1}$ by adding to each component i a term that annihilates the ith component of the residual vector $\mathbf{r}_n$. Using the notation $\mathbf{e}_i = (0,0,\ldots,0,1,0,0,\ldots,0)^T \in \mathbb{R}^N$ for the ith canonical unit vector with 1 at position i, the Jacobi method is of the form

$$\mathbf{x}_n = \mathbf{x}_{n-1} + \alpha_i \mathbf{e}_i \tag{2.1}$$

subject to

$$\mathbf{r}_n^H \mathbf{e}_i = 0 \tag{2.2}$$

for $i = 1,2,\ldots,N$. The value of α_i is determined by the constraint (2.2) as follows. By the definition of the residual vector, an equivalent form of (2.2) is given by

$$(\mathbf{b} - A\mathbf{x}_n)^H \mathbf{e}_i = 0.$$

Inserting (2.1) and using the definition of $\mathbf{r}_{n-1}$ leads to

$$\mathbf{r}_{n-1}^H \mathbf{e}_i = (\alpha_i A \mathbf{e}_i)^H \mathbf{e}_i$$

or, equivalently

$$\alpha_i \mathbf{e}_i^H A \mathbf{e}_i = \mathbf{e}_i^H \mathbf{r}_{n-1}.$$

Assuming that the inverse of $\mathbf{e}_i^H A \mathbf{e}_i = a_{ii}$ exists, we find that

$$\alpha_i = (\mathbf{e}_i^H A \mathbf{e}_i)^{-1} \mathbf{e}_i^H \mathbf{r}_{n-1}. \tag{2.3}$$

In summary, given an update of $\mathbf{x}_n$ in the form of (2.1), the value of α_i is determined by the constraint (2.2) and is given by (2.3). Here, the update and the constraint involve the unit vector $\mathbf{e}_i$ and might be reformulated as

$$\mathbf{x}_n = \mathbf{x}_{n-1} + \operatorname{span}\{\mathbf{e}_i\} \tag{2.4}$$

subject to

$$\mathbf{r}_n \perp \operatorname{span}\{\mathbf{e}_i\}. \tag{2.5}$$

The idea of a projection method to be discussed in the next section is to replace $\operatorname{span}\{\mathbf{e}_i\}$ with more general subspaces.

2.2 General Form of Projection Methods

The approximations $\mathbf{x}_n$ of a projection method are constructed in a suitable subspace $\mathcal{K}$ called the *search subspace*; that is, the iterates are of the form

$$\mathbf{x}_n = \mathbf{x}_{n-1} + \mathcal{K}. \tag{2.6}$$

The actual definition of the iterates is determined by imposing the restriction

$$\mathbf{r}_n \perp \mathcal{L}, \tag{2.7}$$

where $\mathcal{L}$ is another subspace referred to as the *subspace of constraints*. In a general projection method, there are two subspaces, $\mathcal{K}$ and $\mathcal{L}$, rather than a single one as in (2.4) and (2.5). Furthermore, $\mathcal{K}$ and $\mathcal{L}$ are more general than $\operatorname{span}\{\mathbf{e}_i\}$. The iterative scheme defined by (2.6) and (2.7) is called a *projection method onto the search subspace $\mathcal{K}$ along the subspace of constraints $\mathcal{L}$*. A projection method is said to be *orthogonal* if $\mathcal{K} = \mathcal{L}$. In an *oblique* projection method the search subspace is different from the subspace of constraints, i.e., $\mathcal{K} \neq \mathcal{L}$. The general framework of a projection method represents a broad class of iterative methods and still depends on the actual choice of $\mathcal{K}$ and $\mathcal{L}$. Suitable subspaces are identified in the following two lectures. The aim here is to describe the general framework without being specific about the subspaces $\mathcal{K}$ and $\mathcal{L}$.

It is useful to express a projection method in matrix representation. To this end, assume

$$\dim \mathcal{K} = \dim \mathcal{L} = m$$

and let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m$ and $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m$ denote the basis vectors of the subspaces

$$\mathcal{K} = \operatorname{span}\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m\}$$

and

$$\mathcal{L} = \operatorname{span}\{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m\},$$

respectively. Moreover, let

$$V_m = [\mathbf{v}_1\ \mathbf{v}_2\ \cdots\ \mathbf{v}_m]$$

and

$$W_m = [\mathbf{w}_1\ \mathbf{w}_2\ \cdots\ \mathbf{w}_m]$$

denote two $N \times m$ matrices whose columns are the basis vectors of $\mathcal{K}$ and $\mathcal{L}$, respectively. In matrix representation, the iterates of a projection method onto the search subspace $\mathcal{K}$ along the subspace of constraints $\mathcal{L}$ are given by

$$\mathbf{x}_n = \mathbf{x}_{n-1} + V_m \mathbf{y}_m \tag{2.8}$$

where the vector $\mathbf{y}_m \in \mathbb{R}^m$ represents the coefficient vector of $\mathbf{x}_n - \mathbf{x}_{n-1}$ with respect to the basis V_m of $\mathcal{K}$. The vector $\mathbf{y}_m$ is determined by the restriction

$$W_m^H \mathbf{r}_n = \mathbf{0}, \tag{2.9}$$

stating that the residual is orthogonal to all linear combinations of basis vectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_m$ of $\mathcal{L}$.

Using the definition of $\mathbf{r}_n$ and inserting (2.8) into (2.9), the vector $\mathbf{y}_m$ follows from

$$W_m^H(\mathbf{b} - A\mathbf{x}_{n-1} - AV_m\mathbf{y}_m) = \mathbf{0}.$$

Hence, under the assumption that the inverse of $W_m^H A V_m$ exists, we have

$$\mathbf{y}_m = (W_m^H A V_m)^{-1} W_m^H \mathbf{r}_{n-1}. \tag{2.10}$$

Thus, from a conceptual point of view, a general form of a projection method onto $\mathcal{K}$ along $\mathcal{L}$ is given by the scheme depicted in Alg. 2.1. Finally, note the similarity of the structure of (2.10) and (2.3).

$\mathbf{x}_n$ = PROJECTION(A, $\mathbf{b}$, $\mathbf{x}_0$) If $A \in \mathbb{C}^{N\times N}$ and $\mathbf{x}_0$ is a starting vector, this algorithm computes approximations $\mathbf{x}_n$ to the solution of the linear system $A\mathbf{x} = \mathbf{b}$.

1: Choose $\mathbf{x}_0 \in \mathbb{C}^N$
2: **for** $n = 1, 2, 3, \ldots$ **do** {until convergence}
3: Choose subspaces $\mathcal{K}$ and $\mathcal{L}$
4: Choose basis $V_m = [\mathbf{v}_1\ \mathbf{v}_2\ \cdots\ \mathbf{v}_m]$ of $\mathcal{K}$ and
choose basis $W_m = [\mathbf{w}_1\ \mathbf{w}_2\ \cdots\ \mathbf{w}_m]$ of $\mathcal{L}$
5: $\mathbf{r}_{n-1} \leftarrow \mathbf{b} - A\mathbf{x}_{n-1}$
6: $\mathbf{y}_m \leftarrow (W_m^H A V_m)^{-1} W_m^H \mathbf{r}_{n-1}$
7: $\mathbf{x}_n \leftarrow \mathbf{x}_{n-1} + V_m \mathbf{y}_m$
8: **end for**

Figure 2.1: General form of a projection method

2.3 An Optimality Result for $\mathcal{L} = A\mathcal{K}$

In an oblique projection method the search subspace $\mathcal{K}$ is different from the subspace of constraints $\mathcal{L}$. If the search subspace of an oblique projection method is some general subspace $\mathcal{K}$ and the subspace of constraints satisfies $\mathcal{L} = A\mathcal{K}$, the quality of the resulting approximations $\mathbf{x}_n$ can be judged. The crucial issue is that a powerful result is established without explicitly specifying $\mathcal{K}$.

Recall that the goal of any iterative method for the solution of linear systems is to drive the residual vector $\mathbf{r}_n = \mathbf{b} - A\mathbf{x}_n$ to the zero vector. Given a current approximation $\mathbf{x}_{n-1}$, the next approximation $\mathbf{x}_n$ would then ideally minimize the norm of $\mathbf{r}_n$ over all possible candidates

$\mathbf{x} \in \mathbf{x}_{n-1} + \mathcal{K}$ generated by a projection method onto $\mathcal{K}$. Therefore, any iterative scheme that solves the minimization problem

$$\min_{\mathbf{x}\in\mathbf{x}_{n-1}+\mathcal{K}} \|\mathbf{b} - A\mathbf{x}\|$$

in each iteration step n would be an extremely powerful tool for the solution of linear systems. The following result shows that an oblique projection method onto some search subspace $\mathcal{K}$ solves this minimization problem as long as $\mathcal{L} = A\mathcal{K}$.

Theorem 2.1 (Optimality of projection method with $\mathcal{L} = A\mathcal{K}$). *A vector $\mathbf{x}_n$ is the next approximation of a projection method onto the search subspace $\mathcal{K}$ along the subspace of constraints $\mathcal{L} = A\mathcal{K}$ if and only if*

$$\|\mathbf{b} - A\mathbf{x}_n\| = \min_{\mathbf{x}\in\mathbf{x}_{n-1}+\mathcal{K}} \|\mathbf{b} - A\mathbf{x}\|. \tag{2.11}$$

2 Grundlagen moderner iterativer Verfahren

Löse $Ax = b$ iterativ mit Approximationen $x_0, ..., x_n \in C^N$.
Ziel: Residuum $r_n := b - Ax_n \; to0(n \to \infty)$ 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:

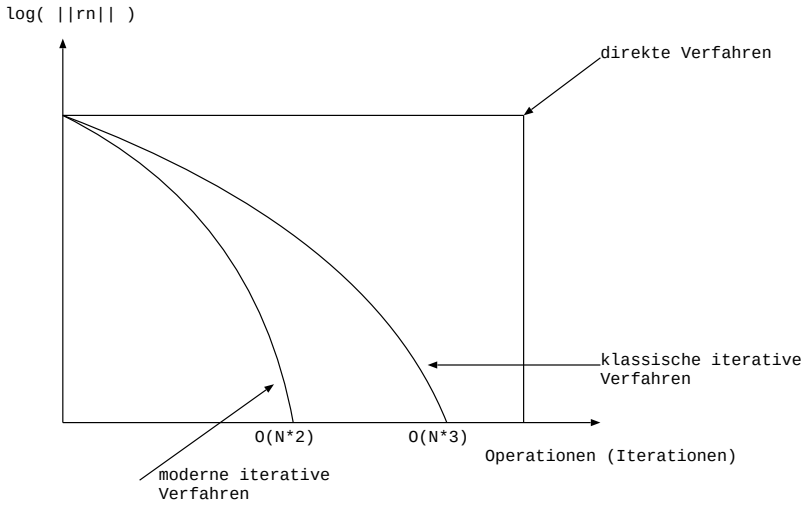
- $||x|| \geq 0$ und $||x|| = 0$ nur für $x = 0$
- $||x+y|| \leq ||x|| + ||y||$ (Dreiecksungleichung)
- $||\alpha * x|| = |\alpha \; *||x||$.

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($a_1, \ldots, a_n$) = $\{y \in C^N | y = Az, \quad z \in C\}$.
$\rightsquigarrow$ 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:

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

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 $r_n = b - Ax_n \perp \mathcal{L}$
$\mathcal{K}$ 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:

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

$$\Leftrightarrow \quad W_m^H \cdot r_{n-1} = \underline{W_m^H \cdot A \cdot W_m \cdot y_m}$$

$$\Leftrightarrow \quad \boxed{y_m = (W_m^H \cdot A \cdot V_m)^{-1} \cdot W_m^H \cdot r_{n-1}}$$

$\rightarrow$ Projektionsverfahren.

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 $x_{n-1} + \mathcal{K}$.

Lecture 3

Construction of a Basis of the Search Subspace (Part I)

$V_n = \text{PowerMethod}(A, \mathbf{v}_1)$ If $A \in \mathbb{C}^{N\times N}$ and $\mathbf{v}_1$ is a suitable starting vector, this algorithm computes a (numerically useless) basis $V_n = [\mathbf{v}_1\ \mathbf{v}_2\ \cdots\ \mathbf{v}_n] \in \mathbb{C}^{N\times n}$ of $\mathcal{K}_n(A, \mathbf{v}_1)$.

1: Choose $\mathbf{v}_1 \in \mathbb{C}^N$ such that $\|\mathbf{v}_1\| = 1$
2: **for** $n = 1, 2, 3, \ldots$ **do** {until invariance}
3: $\quad \widetilde{\mathbf{v}}_{n+1} \leftarrow A\mathbf{v}_n$
4: $\quad \alpha_{n+1} \leftarrow \|\widetilde{\mathbf{v}}_{n+1}\|$
5: $\quad \mathbf{v}_{n+1} \leftarrow \frac{1}{\alpha_{n+1}}\widetilde{\mathbf{v}}_{n+1}$
6: **end for**

Figure 3.1: Power method

Let $\mathbf{e}_n = (0, 0, \ldots, 0, 1, 0, 0, \ldots, 0)^T \in \mathbb{R}^N$ be the nth Cartesian unit vector with 1 at position n.

Theorem 3.1 (Power Method). *In exact arithmetic the vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ generated during the course of Alg. 3.1 form a (poor man's) basis of the Krylov subspace*

$$\mathcal{K}_n(A, \mathbf{v}_1) = \text{span}\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$$

where successive vectors are related by

$$AV_n = V_n L_n + \alpha_{n+1}\mathbf{v}_{n+1}\mathbf{e}_n^T, \tag{3.1}$$

where the recurrence coefficients α_i are collected in the $n \times n$ matrix

$$L_n = \begin{bmatrix} 0 & & & & \\ \alpha_2 & 0 & & & \\ & \alpha_3 & 0 & & \\ & & \ddots & \ddots & \\ & & & \alpha_n & 0 \end{bmatrix}$$

with nonzero elements in the first subdiagonal only.

$V_n = \text{BASICARNOLDI}(A, \mathbf{v}_1)$ If $A \in \mathbb{C}^{N\times N}$ and $\mathbf{v}_1$ is a suitable starting vector, this algorithm computes an orthonormal basis $V_n = [\mathbf{v}_1\ \mathbf{v}_2\ \cdots\ \mathbf{v}_n] \in \mathbb{C}^{N\times n}$ of $\mathcal{K}_n(A, \mathbf{v}_1)$ via the standard Gram–Schmidt process.

1: Choose $\mathbf{v}_1 \in \mathbb{C}^N$ such that $\|\mathbf{v}_1\| = 1$
2: **for** $n = 1, 2, 3, \ldots$ **do** {until invariance}
3: $\quad\widetilde{\mathbf{v}}_{n+1} \leftarrow A\mathbf{v}_n$
4: $\quad$**for** $i = 1, 2, \ldots, n$ **do**
5: $\quad\quad h_{in} \leftarrow \mathbf{v}_i^H \widetilde{\mathbf{v}}_{n+1}$
6: $\quad$**end for**
7: $\quad\widetilde{\mathbf{v}}_{n+1} \leftarrow \widetilde{\mathbf{v}}_{n+1} - \sum_{i=1}^{n} h_{in}\mathbf{v}_i$
8: $\quad h_{n+1,n} \leftarrow \|\widetilde{\mathbf{v}}_{n+1}\|$
9: $\quad\mathbf{v}_{n+1} \leftarrow \frac{1}{h_{n+1,n}}\widetilde{\mathbf{v}}_{n+1}$
10: **end for**

Figure 3.2: Arnoldi method via standard Gram–Schmidt orthogonalization

Let

$$H_n = \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1n} \\ h_{21} & h_{22} & \cdots & h_{2n} \\ & \ddots & \ddots & \vdots \\ & & h_{n,n-1} & h_{nn} \end{bmatrix} \in \mathbb{C}^{n\times n},$$

be an upper Hessenberg matrix; that is, a matrix with zeros below the first subdiagonal.

Theorem 3.2 (Arnoldi). *In exact arithmetic the Arnoldi vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ generated during the course of Alg. 3.2 form an orthonormal basis*

$$V_n^H V_n = I_n$$

of the Krylov subspace

$$\mathcal{K}_n(A, \mathbf{v}_1) = \operatorname{span}\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}.$$

Successive Arnoldi vectors are related by

$$AV_n = V_n H_n + h_{n+1,n}\mathbf{v}_{n+1}\mathbf{e}_n^T, \tag{3.2}$$

and the matrix A is reduced to (full) upper Hessenberg form

$$V_n^H A V_n = H_n \tag{3.3}$$

by means of unitary transformations V_n.

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

Für Gauß-Seidel-Verfahren galt $\mathcal{K}_n = \mathcal{L}_n = span(e_n)$, also orthogonale Projektionsmethode. Aber Resultat von oben nicht anwendbar. Vielleicht hatte man nur schlechtes $\mathcal{K}_n$ gewählt und bei Wahl von "besserem" $\mathcal{K}_n$ erhält man geeignetes Verfahren?

Frage: Welcher Teilraum ist für $\mathcal{K}_n$ geeignet?
Sei $\mathcal{K}_n = span(v_1, \ldots, v_n) = \{V_n y | y \in C^N\}$ wobei $V_n := [v1, \ldots, v_n] \in C^{N\ timesn}$. Dann gilt $x_n = x_y + V_n y_n$ für ein $y_n \in C^N$ und für Grenzwert $n \to \infty$:

$$\begin{aligned} Ax_n = b \quad &\leftrightarrow \quad A \cdot (x_0 + V_n y_n) = b \\ &\leftrightarrow \quad A \cdot V_n \cdot y_n = b - Ax_0 = r_0 \end{aligned}$$

Also: Gesucht ist Basis V_n so, dass $V_n \cdot y_n = A^{-1} \cdot r_0$.

Behauptung: Es gibt kleinsten Index m mit $m <= N$ so, dass Spalten von K_{m+1} linear abhängig sind, d.h. $K_{m+1} \cdot z = 0$ für $z \in C^{m+1}$. Sei $z = (\alpha_0, \ldots, \alpha_m)^T$, dann folgt

$$\begin{aligned} \alpha_0 \cdot r_0 + \alpha_1 \cdot A \cdot r_0 + \ldots + \alpha_m \cdot A^m\ cdotr_0 \quad &= \quad 0 \\ \leftrightarrow \alpha_0 \cdot A^{-1} \cdot r_0 \quad &= \quad -\alpha_1 \cdot r_0 - \alpha_2 \cdot A \cdot r_0 - \ldots - \alpha_m \cdot A^{m-1} \cdot r_0 \end{aligned}$$

Weil m der <u>kleinste</u> Index ist, gilt $\alpha_0 \neq 0 \leftrightarrow A^{-1} \cdot r_0 = K_m \cdot y_m$ mit $y_m = -\frac{1}{r_0} \cdot \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_m \end{pmatrix}$.

Wunsch: $m << N$, dann schnelle Konvergenz!

3.1 Definition

$\mathcal{K}_n(A, r_0) := span r_0, A \cdot r_0, \ldots, A^{n-1} \cdot r_0$ heißt n-ter von A und r_0 erzeugter *Krylov-Teilraum*. Eine Projektionsmethode auf $\mathcal{K}_n = \mathcal{K}_n(A, r_0)$ entlang $\mathcal{L}_n$ heißt *Krylov-Teilraum-Verfahren* (KTV).
Also: in KTV werden $x_n \ldots x_0$ in speziellem Raum, $\mathcal{K}_n(A, r_0)$ kontruiert.

Welche Basis ist für $\mathcal{K}_n(A, r_0)$ geeignet?

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

5 Parallele Programmiermodelle

5.1 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 $\Leftrightarrow$ Aufgabeparallelismus: Parallele Ausführung der gleichen Aufgabe auf verschiedenen Daten $\Leftrightarrow$ parallele Ausführung verschiedener Aufgaben auf gleichen / verschiedenen Daten
- Explizite / implizite Parallelisierung: Programmierer spezifiziert explizit die Aktionen zur parallelen Ausführung $\Leftrightarrow$ Programmierer spezifiziert abstrakte Spezifikation, die Aktionen zur parallelen Ausführung implizieren
- Gemeinsamer $\Leftrightarrow$ verteilter Speicher: Austausch von Daten zwischen verschiedenen Prozessoren über einzelnen gemeinsamen Speicher $\Leftrightarrow$ 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...

5.2 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

5.3 Multithreading uns das Modell des gemeinsamen Speichers

5.4 MPI oder OpenMP?

6 Festlegung des Raumes $\mathcal{L}$

7 Kommunikationszeiten für Basisoperationen von KTV auf Parallelrechnern

8 Modellierung iterativer Verfahren mit der Isoeffi zienzanalyse

8.1 Das Isoeffizienzmodell

Laufzeit von seriellen Algorithmen in Abhängigkeit einer Problemgröße N, also $T_{sqe}(n)$. Bei parallelen Algorithmen zusätzlich Anzahl der Prozessoren p und Architektur des Parallelrechners. Also: $T_{par}(N,p)$.
Das Isoeffizienzmodell setzt Laufzeit des schnellsten bekannten seriellen Algorithmus $T_{seq}(N)$ mit derjenigen Prozessoranzahl in Verbindung, die zur Aufrechterhaltung einer vorgegebenen Effizienz benötigt wird. Ziel ist die Modellierung der *Skalierbarkeit* als Algorithmus auf einer Archtektur, d.h. die Fähigkeit einer zur Prozessoranzahl proportionalen Leistung.

8.2 Definition

Für obiges $T_{seq}(N)$ und $T_{par}(N,p)$ heißen

$$S := \frac{T_{seq}}{T_{par}} \qquad \text{Speedup}$$

und

$$E := \frac{S}{P} \qquad \text{Effizienz}$$

(optimal: $S = p \quad \leftrightarrow \quad E = 1$).

Zwei Beobachtungen:

1. Problemgröße N fest: Steigung $\searrow$ mit $p \nearrow$, d.h. $E \searrow$.
2. Prozessoranzahl p fest: Steigung *nearrow* mit N *nearrow*, d.h. E *nearrow*.

Erwarte, dass $E = const.$, wenn p und N [bzw. $T_{seq}(N)$] "geeignet" erhöht werden. Ein Maß für die Skalierbarkeit ist die Stärke, mit der man dabei N [bzw. $T_{seq}(N)$] erhöhen muss.
Optimaler Speedup wird meist nicht erreicht. Im Isoeffizienzmodell werden alle Gründe für das Nichterreichen unter der Overhead-Funktion

$$T_{over}(N,p) := p \cdot T_{par}(N,p) - T_{ser}(N)$$

zusammengefasst. Es gilt:

$$E := \frac{S}{P} = \frac{T_{seq}}{p \cdot T_{par}} = \frac{T_{seq}}{T_{over} + T_{seq}} = \frac{1}{1 + \frac{T_{over}}{T_{seq}}} \quad (*)$$

8.3 Definition

Ein Paar aus Algorithmus und Architektur heißt ein *skalierbares paralleles System*, falls der Quotient $\frac{T_{over}}{T_{seq}}$ bei adäquater gleichzeitiger Erhöhung von T_{seq} und p konstant bleibt.

Für ein skalierbares paralleles System kann man Wachstumsraten zur Aufrechterhaltung einer konstanten Effizienz aus $(*)$ herleiten, oder äquivalent:

$$1 + \frac{T_{over}}{T_{seq}} = \frac{1}{E} \leftrightarrow \boxed{T_{seq} = \frac{E}{1-E} \cdot T_{over}} \quad (**)$$

Auflösen von $(**)$ nach N als Funktion von p heißt *Isoeffizienzanalyse*.

8.4 Isoeffizienzanalyse eines Iterationsschrittes eines KTV

Annahmen aus letzter VL und s Skalarprodukte / Normen und m Matrix-Vektor-Produkte pro Iterationsschritt. Hier Overhead nur aus Kommunikationszeiten.

$$T_{Comm}^{Ges} = \alpha \cdot \underbrace{T_{Comm}^{SP}}_{=0} + \beta \cdot \underbrace{T_{Comm}^{LK}}_{=0} + m \cdot \underbrace{T_{Comm}^{MVP}}_{\neq 0} + s \cdot \underbrace{T_{Comm}^{Red}}_{\neq 0}$$

$$T_{over}(N,p) = p \cdot m \cdot T_{Comm}^{MVP} + p \cdot s \cdot T_{Comm}^{Red} =$$

$$\begin{cases} \underbrace{2m \cdot t_w \cdot \sqrt{(N)} \cdot p}_{1} + \underbrace{2m \cdot (t_s + t_h) \cdot p}_{2} + \underbrace{2s \cdot (t_s + t_w) \cdot p \cdot \log(p)}_{3} + \underbrace{4s \cdot t_h \cdot p^{\frac{3}{2}}}_{4} & \text{für Blockd.v.} \\ \underbrace{4m \cdot t_w \cdot \sqrt{(Np)}}_{5} + \underbrace{4m \cdot (t_s + t_h) \cdot p}_{6} + \underbrace{2s \cdot (t_s + t_w) \cdot p \cdot log(p)}_{3} + \underbrace{4s \cdot t_h \cdot p^{\frac{3}{2}}}_{4} & \text{für Blockzyklisched.v.} \end{cases}$$

Aus Annahmen folgt: $T_{seq}(N) = c \cdot N \cdot t_a$, wobei $c = const.$ und t_a Zeit für arithmetische Operationen in C.

8.4.1 Blockdatenverteilung

1. $c \cdot B \cdot t_a = \frac{E}{1-E} \cdot 2m \cdot t_w \cdot \sqrt{(N)} \cdot P$
   $\leftrightarrow N = (\frac{2m \cdot t_w \cdot E}{c - t_a \cdot (t-E)}^2 \cdot p^2 \rightarrow O(p^2)$
2. $c \cdot N \cdot t_a = \frac{e}{1-E} \cdot 2m \cdot (t_s + t_h) \cdot p$
   $\rightarrow N = O(p)$
3. $c \cdot N \cdot t_a = \ldots\ p \cdot \log(p)$
   $\rightarrow N = O(p \cdot \log(p))$
4. $c \cdot N \cdot t_a = \frac{E}{1-E} \cdot 4s \cdot t_h \cdot p^{\frac{3}{2}}$
   $\leftrightarrow N = \frac{4x \cdot t_n \cdot E}{c \cdot t_a \cdot (1-E)} \cdot p^{\frac{3}{2}} \rightarrow O(p^{\frac{3}{2}})$

$\rightarrow N = O(p^2)$.

8.4.2 Blockzyklischedatenverteilung

1. (5) $c \cdot N \cdot t_a = \frac{E}{1-E} \cdot 4m \cdot t_w \cdot \sqrt{(N)} \cdot sqrt(p)$
   $\leftrightarrow N = (\frac{4m \cdot t_w \cdot E}{c \cdot t_a \cdot (1-E)}^2 \cdot p \rightarrow O(p)$
2. (6) wie (2)
   $\rightarrow N = O(p)$
3. (3) wie oben
4. (4) wie oben

$\rightarrow N = O(p^{\frac{3}{2}})$.

Fazit: Datenverteilung ist wichtig für Skalierbarkeit.

8.5 Anforderungen an parallelen Algorithmenentwurf

In Blockzyklischerdatenverteilung: $N = f(p)$, wobei t_h, t_a, c, E fest. Einziger Entwurfsparameter ist $s =$ Anzahl von Skalarprodukten / Normen. Bisher modelliert Hintereinanderausführung von s Skalarprodukten. Was wäre, wenn alle s Skalarprodukte voneinander unabhängig sind?
$\rightarrow$ Also führe nicht s-mal Reduktion auf skalaren Daten (l = 1) durch, sondern 1-mal Reduktion auf Vektoren der Länge s.
In obiger Isoeffizienzanalyse wird der Ausdruck für s Reduktionen mit (l = 1):

$$s \cdot T_{Comm}^{Red} = 2s \cdot [(t_s + t_w) \cdot \log(p) + 2 \cdot t_h \cdot \sqrt{(p)}]$$

durch eine Vektorreduktion mit (l = s) ersetzt:

$$s \cdot T_{Comm}^{Red} = 2 \cdot [(t_s + s \cdot t_w) \cdot \log(p) + 2 \cdot t_h \cdot \sqrt{(p)}]$$

8.6 Satz

$N = O(p^{\frac{3}{2}})$.

9 Skalierbare Lanczos-Algorithmen

Aus Isoeffizienzanalyse: Vermeide globale Kommunikation, die durch Reduktionen (innere Produkte, Normen) verursacht wird, dadurch, dass Reduktionen eines Iterationsschrittes unabhängig voneinander sind. Betrachte deshalb Lanczos-Alg. (4.1) zur Erzeugung von Krylov-Teilräumen $\mathcal{K}_n(A, v_1)$ und $\mathcal{K}_n(A^H, w_1)$ in traditioneller serieller Form:

10 Paralleles Matrix-Vektor-Produkt

Basisoperationen in KTV: für gegebenes $x \in R^N$ und große, dünnbesetzte Matrix $A \in R^{N \times N}$ berechne $y = Ax$ auf p Prozessoren. Unter Ausnutzung der Dünnbesetztheit gilt für die i-te Komponenten von y:

$$y_i = \sum_{j=1, a_{ij} \neq 0}^{N} a_{ij} \cdot x_j$$

Graphinterpretation einer dünnbesetzten Matrix mit symmetrischem non-zero Muster:
Assoziiere einen ungerichteten Graphen G_A mit $A : G_A = (V, E)$
Knotenmenge $V = 1, 2, \ldots, N$
Kantenmenge $E = (i, j) | a_{ij} \neq 0, i \neq j$.

$$y_i = \sum_{j=1, (i,j) \in E}^{N} a_{ij} \cdot x_j$$

Konsistente Datenverteilung:
x_i, y_i und Zeile i, d.h. $a_{ij} \neq 0$ für $j = 1, \ldots, N$ auf demselben Prozessor. Beschreibe Datenverteilung durch Abbildung $P : V \to 1, 2, \ldots, p$ die die Knoten von G in p disjunkte Zusammenhangskomponenten V_i mit $V = \bigcup_{i=1}^{p} v_i$ aufteilt.
Abbildung P nennt man p-Partition. Eine 2-Partition heißt *Bisektion*.

10.1 Balancierung der Rechenlast

Anz. Multiplikationen zur Berechnung von y_i = $degree(i) + 1$
Anz. Additionen zur Berechnung von y_i = $deg(i)$
Gewichte Knoten i mit Anz. seiner auszuführenden Operationen: $\varphi(i) = 2 \cdot deg(i) + 1$
Definiere Summe aller Knotengewichte einer Zusammenhangskomponente V_i $\varphi(V_i) = \sum_{j \in V_i} \varphi(j) = \sum_{P(j)=i} \varphi(j)$ modelliert Anz. Operationen auf Prozessor i.
Eine p-Partition heißt balanciert, falls $\varphi(v_i) <= S'$ für $i = 1, \ldots, p$, wobei S' kleinste Schranke ist, die ganzzahliges Vielfaches von $\max_i \varphi(i)$ ist, mit $\frac{\sum_{j=1}^{p} \varphi(v_j)}{p} <= S'$.

Abb. 103: 3-Partition ist balanciert, weil $\varphi(i) = 2 \cdot 3 + 1 = 7 \quad \forall i$.
$\varphi(v_1) = 4 \cdot 7 = 28$ Rechenlast in Komponente 1
$\varphi(v_2) = \varphi(v_3) = 3 \cdot 7 = 21$
$\frac{70}{3} = 23, \ldots <= 28, \quad \varphi(v_i) <= 28 \quad \forall i$.
(Ähnlich bei inhomogenen Architekturen, bloss Einführung unterschiedlicher Gewichte)

10.2 Minimierung der Kommunikation

11 SPAI-Konditionierung

11.1 Vorkonditionierung

Außer für (full) GMRES weiß man nichts über Konvergenzverhalten von KTV für nichtsymmetrische Probleme. Um Konvergenz zu gewährleisten oder zu beschleunigen, wird gegebenes System $Ax = b$ in ein *vorkonditioniertes* System

$$A \cdot M \cdot y = b \quad \text{und} \quad x = M \cdot y \qquad \text{(rechts-vorkonditioniert)}$$

oder

$$M \cdot A \cdot x = M \cdot y \qquad \text{(links-vorkonditioniert)}$$

transformiert. Konvergenzbeschleunigung, falls $A \cdot M$ (rechts) bzw. $M \cdot A$ (links) eine gute Approximation an Identität I ist. Zusätzliche Arbeit für Vorkonditionierung:

1. einmalige Berechnung von M
2. Lösen von Gleichungssystemen mit Koeffizientenmatrix M in jeder Iteration (implizite Vorkonditionierung) oder Matrix-Vektor-Produkt mit M in jeder Iteration (explizite Vorkonditionierung).

11.2 Anforderungen an M

- Lösen von $Mu = v$ $(u, v \in C^N, M \in C^{N \times N}$ einfach. Extremfall: $M = I$ (aber dann keine Konvergenzbeschleunigung).
- Hohe Konvergenzbeschleunigung. Extremfall: $M = A^{-1}$ (dann Konvergenz im ersten Schritt, aber genauso aufwendig zu Lösen wir Ursprungssystem).

Nützliche Vorkonditionierer M liegen zwischen diesen beiden Extremfällen. Kriterium für die Güte eines Vorkonditionieres M ist die Zeit, nicht die Anzahl von Iterationen.
Es gibt eine Fülle von unterschiedlichen Klassen von Vorkonditionieren.

11.3 SParse Approximate Inverse (SPAI) Vorkonditionierer

Betrachte Rechts-Vorkonditionierung mit Ziel $A \cdot M \approx I$.
Viele Vorkonditionierer sind zwar mächtig, aber inherent seriell. Ziel von SPAI: Konstruktion von M parallel und Matrix-Vektor-Multiplikation mit M parallel (SPAI ist explizite Vorkonditionierung).

Idee: Minimiere $||A \cdot M - I||$, aber welche Norm?
Sei $B = [b_1, \ldots, b_n] \in R^{m \times n}$.

$$||B||_F := (\sum_{i=1}^{m} \sum_{j=1}^{n} |b_{ij}|^2)^{\frac{1}{2}} = (\sum_{j=1}^{n} ||b_j||_2^2)^{\frac{1}{2}} \quad \text{Frobenius-Norm}$$

Minimierung von $||A \cdot M - I||$ in 1,2,∞-Normen zu aufwendig. Minimiere stattdessen in Frobenius-Norm.

$$||A \cdot M - I||_F^2 = \sum_{j=1}^{N} ||(A \cdot M - I) \cdot e_j||_2^2 \qquad (*)$$

mit $e_j = (0,\ldots,0,1,0,\ldots,0)^T$. Sei $M = [m_1,\ldots,m_N]$, dann zerfällt das Minimierungsproblem $\min_{M \in C^{N\times N}} ||A \cdot M - I||_F$ in N unabhängige Least-Squares Probleme

$$\min_{m_j \in C^N} ||A \cdot m_j - e_j||_2 \quad (j = 1,\ldots,N) \qquad (**)$$

Bei expliziter Vorkonditionierung ist Matrix-Vektor-Produkt mit M effizient, also M dünnbesetzt.

11.4 Annahme: Festes Sparsity Pattern von M

Betrachte k-tes Problem $(**)$.

Lösung des $|J| \times |J|$ Least-Squares Problem über QR-Zerlegung.
Setze $\overline{r_k} = A(\cdot, J) \cdot \overline{m_k} - e_k = A\overline{m_j} - e_k$ Residuum von Least-Squares Problem. Falls $\overline{r_k} = \underline{0}$, ist m_k die k-te Spalte von A^{-1} und man kann keine Verbesserung erzielen. Da m_k sparse, ist im Allgemeinen $\overline{r_k} \neq 0$.

11.5 Annahme: Fortschreitende Verbesserung des Sparsity Pattern

Starte mit gegebenem Sparsity Pattern (z.B. M diagonal) und versuche Pattern so auszudehnen, dass $\overline{r_k}$ reduziert wird. Abbruch, falls $||\overline{r_k}||_2 \leq \varepsilon$ für $k = 1,\ldots,N$. Parameter ε steuert, wie genau $A \cdot M \approx I$ (und wie hoch Anz. non-zeros in M ist).

- Kleines ε: hohe Kosten zur Berechnung von M, dichtes M, große Konvergenzbeschleunigung.
- Großes ε: Gerine Kosten zur Berechnung von M, sparse M, geringe Konvergenzbeschleunigung.

In welchen Komponenten kann $\overline{r_k}$ überhaupt reduziert werden?
$\mathcal{L} := l|$ l-ter Eintrag von $\overline{r_k} \neq 0$.
Welche Spalte von A steuert dies?
Für jedes $l \in \mathcal{L}$ setze $\mathcal{N}_{\updownarrow} := n|A(l,n) \neq 0$ und $n \notin \mathcal{L}$.
Was sind potentielle neue Kandidaten für Pattern Erweiterung?
$\overline{J} := \bigcup_{l\in\mathcal{L}} \mathcal{N}_{\updownarrow}$.

11.5.1 Beispiel

$J = 4 \qquad \mathcal{L} = 1,3,4,5$
$\mathcal{N}_{\infty} = 2 \quad \mathcal{N}_{\ominus} = 5 \quad \mathcal{N}_{\triangle} = 2,3 \quad \mathcal{N}_{\triangledown} = 2,5$
$\overline{J} = 2,3,5$
Schätze mit einfache Rechnung ab, welche j aus $\overline{J}$ das Residuum $\overline{r_k}$ am meisten reduzieren $\rightsquigarrow$ *SPAI*.

12 Bipartite Graphpartitionierung

12.1 A dünnbesetzt, symmetrisches nonzero pattern, konsistente Datenverteilung

a)
Konsistente Datenhaltung: Zeile i, x_i und y_i alle auf Prozessor $P(i)$.

$$y = \sum_{i=1}^{N} y_i \cdot e_i \qquad e_i \text{ i-ter kanonischer EH-Vektor aus } R^N$$

$$y_i = \sum_{j=1, a_{ij}\neq 0}^{N} a_{ij} \cdot x_j \qquad \text{Summe über alle Zeilen} \rightarrow \text{Knoten } i \text{ repräsentiert Zeile } i$$

$$y_i = a_{ii} \cdot x_i + \sum_{(i,j)\in E} a_{ij} \cdot x_j$$

$$= \underbrace{a_{ii} \cdot x_i + \sum_{(i,j)\in E, P(i)=P(j)} a_{ij} \cdot x_j}_{\text{lokal auf Proz. P(i), weil Zeile i und } x_j \text{ mit P(j) = P(i) auf selbem Proz.}} + \underbrace{\sum_{(i,j)\in E, P(i)=P(j)} a_{ij} \cdot}_{x_j \text{ muss von P(j) nach P(i) ges}}$$

D.h.: Kommuniziere Daten für Multiplikationen, Additionen lokal.

b)
Konsistente Datenhaltung: Spalte j, x_j und y_j auf Prozessor $P(j)$.

$$y = \sum_{j=1, a_{ij}\neq 0}^{N} y^{(j)} \qquad y^{(j)} \in R^N \text{ ist j-te Spalte von A multipliziert mit } x_j$$

$$y^{(j)} = \underbrace{a_{jj} \cdot x_j \cdot e_j + \sum_{(i,j)\in E} a_{ij} \cdot x_j \cdot e_i}_{\text{lokal auf Prozessor P(j), weil Spalte j und } x_j \text{ auf selbem Prozessor}}$$

$$y_j = \underbrace{a_{jj} \cdot x_j + \sum_{(i,j)\in E, P(i)=P(j)} y_i^{(j)}}_{lokal} + \underbrace{\sum_{(i,j)\in E, P(i)\neq P(j)} y_j^{(i)}}_{y_j^{(i)} \text{ muss von P(i) nach P(j) geschickt werden}}$$

D.h.: Kommuniziere Daten für Addition, Multiplikation lokal.

Fazit: Für symmetrische nonzero pattern ist Graphpartitionierungsproblem unabhängig von Zeilen- oder Spaltenverteilung der Daten.
Datenlokalität kann erhöht werden durch Umnummerierung der Knoten im Graph, ohne Kanten zu verändern. In Matrixnotation wird dies durch eine sog. *symmetrische Permutation* beschrieben: $A \rightarrow P^T AP$ mit $P^T P = PP^T = I$. Bestimme P durch Graphpartitionierung und löse statt $Ax = b$ nun $(P^T AP)(P^T x) = P^T b$. Das meint: Variablenumnummerierung und Umsortierung der Zeilen in gleicher Art und Weise.

12.2 A dünnbesetzt, nicht-symmetrisches nonzero pattern, konsistente Datenverteilung

a)
Wende Graphpartitionierung auf $A + A^T$ an, dann wieder symmetrisches nonzero pattern. Aber nur gut für “fast” symmetrische nonzero pattern. Oft schlecht.

b)

12.3 A dünnbesetzt, nicht-symmetrisches nonzero pattern, konsistente Datenverteilung, explizite Vorkonditionierung