8 Lineare Algebra für dünnbesetzte Matrizen

8.1 Tridiagonale Gleichungssysteme

T. Leighton, *Introduction to Parallel Algorithms and Architectures*, Morgan Kaufmann, 1992. Abschnitt 1.3.3.

W. H. Press, S.A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes in C*, Cambridge University Press, 2. Ausgabe, 1992. Abschnitt 2.4.

$$A := \begin{pmatrix} d_1 & u_1 & & & \\ l_2 & d_2 & u_2 & & \\ & \ddots & \ddots & \ddots & \\ & & l_{n-1} & d_{n-1} & u_{n-1} \\ & & & l_n & d_n \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

A sei symmetrisch und positive definit ($\forall v : v \cdot A \cdot v > 0$) oder diagonaldominant ($|d_j| > |l_j| + |u_j|$).

8.1.1 Gausselimination

- Speicher $5n$ Zahlen
- n Divisionen (extra zählen)
- $3n$ Subtraktionen, $4n$ Multiplikationen
- Kein Pivoting nötig
- Inhärent sequentiell

8.1.2 Odd-Even-Reduktion

- Substituiere
$$x_i = \frac{1}{d_i}(b_i - l_i x_{i-1} - u_i x_{i+1})$$
für ungerade i (mit $x_0 = x_{n+1} = 0$)
- Setze in Gleichungen für gerade i ein
- Löse rekursiv das System der Größe $\lfloor n/2 \rfloor$
- Rücksubstitution
- Deutlich mehr arithmetische Operationen als Gausselimination ($\approx 17n$ Multiplikationen)

Lösung über eine lineare Rekurrenz

$$\begin{pmatrix} x_{i+1} \\ x_i \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} -\frac{d_i}{u_i} & -\frac{l_i}{u_i} & \frac{b_i}{u_i} \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{G_i} \begin{pmatrix} x_i \\ x_{i-1} \\ 1 \end{pmatrix}$$

also

$$\begin{pmatrix} x_{i+1} \\ x_i \\ 1 \end{pmatrix} = \prod_{j=i}^{1} G_i \begin{pmatrix} x_1 \\ 0 \\ 1 \end{pmatrix}$$

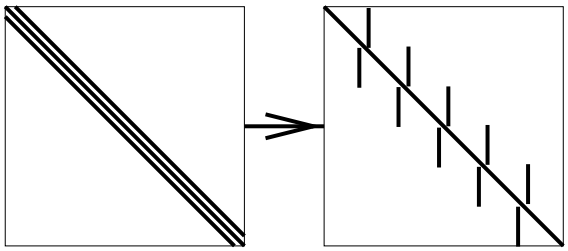
- Berechnung mittels paralleler Postfixberechnung (wie auf Präfix umleiten?)
- Beispiel für nichtassoziative Operation
- Oft numerisch instabil
- Ebenfalls viele arithm. Operationen
- Lösung durch 3 aufeinanderfolgende Rekurrenzberechnungen für 2×2 Matrizen ist besser. (Parallele LU-Zerlegung)

MPI: Präfixsummen

`MPI_Scan` ähnelt `MPI_Reduce`. Nur wird nicht ein einziges Ergebnis produziert, sondern auf PE i wird $a_0 \otimes a_1 \otimes \cdots \otimes a_i$ bestimmt. Der Prototyp ist

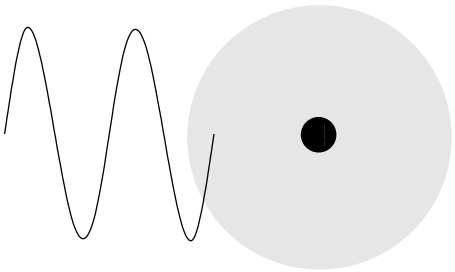
```
int MPI_Scan(void *sendbuf,
             void *recvbuf, int count,
             MPI_Datatype datatype,
             MPI_Op op,
             int root, MPI_Comm comm);
```


8.1.3 Lokale Problemreduktion



- $n \times n \to 2P \times 2P$
- Etwas effizienter für große n ($\approx 17n$ Flops)

8.2 Anwendungsbeispiel: 2D zeitabhängige Dirac-Gleichung



- 10^{19} W/cm^2, Elektron erreicht relativistische Geschwindigkeiten
- 2D um elektrisches und magnetisches Feld zu erfassen
- Hohe Auflösung um de Broglie Wellenlänge des Elektrons zu erfassen
- $\to$ kleine Zeitschritte
- Viele Zeitschritte um "niedrige" Schwingungsfrequenz des Lichts zu erfassen

8.2.1 Die partielle Differentialgleichung

$$\mathrm{i}\frac{\partial}{\partial t}\Psi(x,z;t) =$$
$$\left\{c\alpha\cdot\left[\mathbf{p}-\frac{1}{c}\mathbf{A}(x,z;t)\right]+c^2\beta+U(x,z)\right\}\Psi(x,z;t)$$

8.2.2 Numerischer Ansatz

- Räumliche und zeitliche Diskretisierung
- Differentialoperatoren → Differenzenoperatoren
- Alternating direction implicit method
- Koeffizienten sind 4×4 komplexe Matrizen mit spezieller Struktur
- Problem in z-Richtung hat identische a priori bekannte Koeffizienten → weniger Arithmetik

8.2.3 Die Differenzengleichung

$$\begin{aligned}
&\frac{1}{\Delta t}[\Psi^{t+1/2}(j,k) - \Psi^t(j,k)] = \\
&\quad -\frac{c}{2}(\alpha_x^{(u)}\Delta_j^+ + \alpha_x^{(l)}\Delta_j^-)\Psi^{t+1/2}(j,k) \\
&\quad -\frac{c}{2}(\alpha_z^{(u)}\Delta_k^+ + \alpha_z^{(l)}\Delta_k^-)\Psi^{t}(j,k) + \\
&\quad +\frac{i}{2}[\alpha_z A^t(j,k) - \beta c^2 + V(j,k)]\Psi^t(j,k), \\
&\frac{1}{\Delta t}[\Psi^{t+1}(j,k) - \Psi^{t+1/2}(j,k)] = \\
&\quad -\frac{c}{2}(\alpha_x^{(u)}\Delta_j^+ + \alpha_x^{(l)}\Delta_j^-)\Psi^{t+1/2}(j,k) \\
&\quad -\frac{c}{2}(\alpha_z^{(u)}\Delta_k^+ + \alpha_z^{(l)}\Delta_k^-)\Psi^{t+1}(j,k) + \\
&\quad +\frac{i}{2}[\alpha_z A^{t+1}(j,k) - \beta c^2 + V(j,k)]\Psi^{t+1}(j,k)
\end{aligned}$$

Zeilenweise Zerlegung

+ Schwieriger Schritt lokal
- Leichter Schritt: n unabhängige block-tridiagonale LGS

Transposition

+ Alles lokal → wenig Arithmetik
– Seeehr viel Kommunikation

2D-Zerlegung

+ Weniger Kommunikation
– Alle Systeme müssen durch verteilten Alg. gelöst werden

8.2.4 m unabhängige Systeme

All-to-all Kommunikation:

- PE i sammelt die reduzierten Systeme $in/P \leq j < (i+1)n/P$
- $T_{\text{comm}} = T_{\text{all-to-all}}(\mathrm{O}(n/P), P)$

Parallele Lösung der Systeme

- In der Literatur beschriebene Verfahren $T_{\text{comm}} = \mathrm{O}((T_{\text{start}} + nT_{\text{byte}}) \log P)$
- Mit Pipelining $T_{\text{comm}} = \mathrm{O}(T_{\text{start}} \log P + nT_{\text{byte}})$

– Kompliziert

Sequentielle Lösung der Einzelsysteme

+ Optimale Anzalh arithm. Operationen
+ Ergebnis auf jeden Fall unabh. von Prozessorzahl
- Chunks von k Systemen auf einmal
  - Startups bis Ergebnisse eintrudeln: $4PT_{\text{start}} \ll n^2 T_{\text{coeff}}/P$, d.h. $n \gg 2PT_{\text{start}}/T_{\text{coeff}}$
  - Berechnungen bis Ergebnisse eintrudeln $knT_{\text{coeff}} \ll n^2 T_{\text{coeff}}/P$, d.h. $k \ll n/P$
  - Gesamtzahl Startups $4nT_{\text{start}}/k \ll n^2 T_{\text{coeff}}/P$, d.h. $k \gg \frac{PT_{\text{start}}}{nT_{\text{coeff}}}$

Verallgemeinerungen

- Diagonalen bestehen selbst aus Matrizen (blocktridiagonale Systeme) (arithm. Aufwand?)
- k Diagonalen (arithm. Aufwand?)
- Bandstruktur muss erst gefunden werden (i.allg. NP vollst)
- d-D ADI
- Matrizen ohne Bandstruktur: Problem ist Vermeidung von **fill in**

8.3 Matrix-Vektor-Multiplikation

- Anzahl nicht-null Elemente oft in $O(n)$
- Repräsentation der Matrix
- Grundprinzip der Parallelisierung: Jedes PE bekommt Teilmenge der Zeilen
- `MPI_Allgather` des Vektors oft zu teuer
- Idee: Vektor nur dahin schicken wo gebraucht
- $\rightarrow$ Lokalitätserhaltende Verteilung wichtig

8.4 Iterative Gleichungslöser

8.4.1 Gauss-Seidel Iteration

Idee: Wenn $x(k)$ eine gute Näherung ist, dann ist

$$x_i(k+1) := \frac{1}{a_{ii}}(b_i - \sum_{j \neq i} a_{ij} x j)$$

eine Verbesserung.

- Paralleler Update für *unabhängige* Indizes aus einer Menge C mit $\forall i,j \in C : a_{ij} = 0$
- Um alle Werte zu verbessern benötigt man **Färbung** des Adjazenzgraphen $G = (\{1,\ldots,n\}, \{(i,j) \mid a_{ij} \neq 0\})$
- Bei regelmäßigen Matrizen, z.B. Gittern ist eine optimale Färbung oft leicht zu finden
- Abbruchbedingung: **Residuum** $r(k) := b - Ax_k$ ist klein. Zum Beispiel $\|r(k)\|_2 \leq \epsilon \|r(0)\|_2$

8.4.2 Successive Overrelaxation (SOR)

Gewichtetes Mittel zwischen altem und neuem Wert

$$x_i(k+1) := (1-\omega)x_i(k) + \frac{\omega}{a_{ii}}(b_i - \sum_{j \neq i} a_{ij} xj)$$

ggf. schnellere Konvergenz. Vor allem für $\omega > 1$!

8.4.3 Konjugierte Gradienten

- Für symmetrische positiv definite Matrizen
- Verallgemeingerung: Bikonjugierte Gradienten
- Läßt sich auch als direkter Löser auffassen: (maximal n Iterationen)
- **Präkonditionierung:** Verbessere Konvergenz durch Transformation $\tilde{A} = RAR^T$

8.4.4 Multi-Grid Methoden

- Arbeitet mit sukzessiven Vergröberungen des Adjazenzgraphen
- Operationen
  - Relaxation (z.B. Gauss-Seidel)
  - Projektion fein→grob
  - Interpolation grob→fein
- Interpolierte Lösung eines groben Systems kann auch zur Präkonditionierung des feinen Systems verwendet werden.
- In vielen Fällen Ausführungszeit $O(n)$!
- **algebraische multi-grid**: Eingaben ohne geometrische Interpretation

Blick über Tellerrand

- Adaptive Gitterverfeinerung
- Unstrukturierte Gitter
- Linearizierung nichtlinearer partieller Differentialgleichungen
- Berechnung von Eigenwerten und Eigenvektoren
- Fast Fourier Transformation