

Zusammenfassung WR

Samuel Brinkmann (Matr. 624568)

5. Februar 2023



Inhaltsverzeichnis

Inhaltsverzeichnis	2
1 Vorlesung I	3
1.1 Einführung	3
1.2 Modellierung	4
1.3 mathematisches Modell zu numerischem Modell	4
2 Vorlesung II	5
2.1 Finite Differenzen in 2D	5
2.2 LinA Wiederholung	6
3 Vorlesung III	7
3.1 Speicher	7
3.2 Dichte Vektor-Matrix-Operationen	7
3.3 Dünn besetzte Matrizen	8
4 Vorlesung IV	9
4.1 direkte Lösungsverfahren	9
4.2 Cholesky-Zerlegung	9
4.3 Iterative Lösungsverfahren	10
5 Vorlesung V	11
5.1 Konjugierte Gradienten (CG)	11
5.2 Parallelität	12
5.3 Performance-Kriterien	13
5.4 Parallele Basiskonzepte	14
6 Vorlesung VI	15
6.1 Parallele Systeme mit gemeinsamem Speicher	15
6.2 Parallele Systeme mit verteiltem Speicher	15
6.3 MPI (Message Passing Interface)	16

Kapitel 1

Vorlesung I

1.1 Einführung

Def.: Lösen von mathematisch formulierten Problemstellungen die physikalische Sachverhalte beschreiben mit dem Computer.

Wissenschaftlicher Rechnen \iff Rechnergestützte (Ingenieur-)Wissenschaft

Entwurf von Verfahren \iff Verwendung dieser Verfahren

Workflow:

physikalisches Problem

\implies mathematisches Modell

\implies numerisches Modell

\implies Lösungsmethode, Code, Rechner (Fokus in WR)

Problemstellungen:

1 Agentenbasierte Simulation

2 (Nicht-)Lineare Optimierung

3 Reduktion von Sensordaten

Beispiel Anwendungen:

1 Wetter und Klima

2 Computertomografie

3 Simulationen von Pandemien



1.2 Modellierung

Def.: Ein Modell ist ein vereinfachtes Abbild der Realität.

Phasen: (Beispiel Pandemie Simulation)

- 1 Abgrenzung: Irrelevantes ausschließen
(Lieblingsfarbe, Schuhgröße)
- 2 Reduktion: Irrelevantes ausschließen
(Kontaktmatrix statt positionsgetreue Nachverfolgung)
- 3 Dekomposition: Bildung einzelner Segmente
(Populationsgruppen nach Region, Alter, Impfstatus, ...)
- 4 Aggregation: Zusammenfassung von Segmenten
(nur jeden 1000. Menschen simulieren)
- 5 Abstraktion: Bildung von Klassen
(S (susceptible), I (infected), R (recovered))

1.3 mathematisches Modell zu numerischem Modell

Mathematisches Modell: System von PDEs

—> Gleichungen geben nur Kopplung der Variablen an und keine tatsächlichen Werte

Lösung: Ableitungen werden durch den Differenzenquotienten approximiert

$$\left. \frac{\partial f}{\partial x} \right|_{x=x_0} \longrightarrow \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

Diskretisierung: Konstruktion eines Gitternetzes, wobei jede Variable an ihre benachbarten Gitterpunkte gekoppelt ist. So wird aus dem System von PDEs ein System algebraischer Gleichungen (endlich viele).



Kapitel 2

Vorlesung II

2.1 Finite Differenzen in 2D

FD-Approximation der Wärmeleitung in 2D

Wärmeleitungsgleichung:

$$\Delta u = \frac{\partial u}{\partial t} \text{ mit } \Delta = \sum_{i=1}^{\#dim} \frac{\partial^2}{\partial x_i^2}$$

Randbedingungen:

- a linker Rand besitzt konstanten Wert u_0
- b rechter Rand besitzt konstanten Wert u_1

Approximation durch 2D-Gitter $(x_j, t_n)_{j=1, \dots, N}$ mit $x_0 = 0$ und $x_N = 1$:

$$\begin{aligned} u_0^n &= u_0 \quad \forall n \quad (\text{Randbedingung}) \\ u_N^n &= u_1 \quad \forall n \quad (\text{Randbedingung}) \\ u_j^0 &= f(x_j) \quad \text{für } 1 < j < N \quad (\text{Anfangswerte}) \\ \frac{\partial u}{\partial t} &\approx \frac{u_j^{n+1} - u_j^n}{\Delta t} \\ \frac{\partial^2 u}{\partial x^2} &\approx \frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2} \quad (\text{explizit}) \\ \frac{\partial^2 u}{\partial x^2} &\approx \frac{u_{j-1}^{n+1} - 2u_j^{n+1} + u_{j+1}^{n+1}}{\Delta x^2} \quad (\text{implizit}) \end{aligned}$$

In PDE einsetzen und nach u_j^{n+1} umstellen!



5-Punkte-Stern für 2D

- 1 Randpunkte: Randbedingung
- 2 Innere Punkte:
 - a) Randfern: 5-Punkte-Stern
 - b) Randnah: Anpassung durch Einfügen der Randbedingung für Randknoten
- 3 LGS: $A_h u_h = q_h$ mit $q_h = -\hat{A}_h g + f$ (g Randwerte, f 5-Punkte-Stern)

$$(\hat{A}_h)_{ij} = -\frac{1}{h^2}, \text{ falls Knoten } i \text{ randnah und } j \text{ ein Nachbar mit 5-Punkte-Stern ist}$$

$$(\hat{A}_h)_{ij} = 0, \text{ sonst}$$

$$f_{ij} = \frac{1}{h^2}(-u_{i,j-1} - u_{i-1,j} + 4u_{i,j} - u_{i+1,j} - u_{i,j+1})$$

2.2 LinA Wiederholung

Laplace-Matrix für Graphen: $L = \text{Gradmatrix} - \text{Adjazenzmatrix}$

geometrische Bedeutung LGS: Schnittmenge von $\#dim$ Hyperebenen

Matrixnorm: $\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}$

Kondition einer Matrix: $\text{cond}(A) = \|A\| \|A^{-1}\|$

Kondition einer sym. Matrix: $\text{cond}(A) = \frac{|\lambda_{max}|}{|\lambda_{min}|}$



Kapitel 3

Vorlesung III

3.1 Speicher

Zeitliche Lokalität: Adressbereiche, auf die zugegriffen wird, werden auch in naher Zukunft mit hoher Wahrscheinlichkeit wieder benutzt.

Beispiel: `int sum`

```
sum = 0
for i in range(1, n):
    sum += 1
```

Räumliche Lokalität: Nach einem Zugriff auf einen Adressbereich erfolgt mit hoher Wahrscheinlichkeit der nächste Zugriff auf eine Adresse in unmittelbarer Nachbarschaft.

Beispiel: `list A`

```
A = [...]
for i in range(len(A)):
    print(A[i])
```

ACHTUNG: Aufpassen, ob Arrays in *column-major order* oder *row-major order* im Speicher liegen (räumliche Lokalität).

3.2 Dichte Vektor-Matrix-Operationen

Skalarprodukt $a^T \cdot b$: $\mathcal{O}(n)$

Skalarprodukt $a \cdot b^T$: $\mathcal{O}(n)$

Matrix-Vektor-Produkt $A \cdot b$: $\mathcal{O}(n^2)$

Matrix-Matrix-Produkt $A \cdot B$: $\mathcal{O}(n^3)$ (Cache-Effizient durch Blockbasierte Abarbeitung)



3.3 Dünn besetzte Matrizen

CSR - compressed sparse row

3 Arrays der Datenstruktur:

- IR: Größe $N+1$, Index des Zeilenstarts in anderen Arrays
- JC: Größe nnz , Spaltenindex des Eintrags
- NUM: Größe nnz , Wert des Eintrags

Alternativen: CSC, CSR mit Diagonalverschiebung (Diagonale in separatem Array)

Grundoperationen:

Zeilenindizierung $A[i]$: $\mathcal{O}(1)$

SpMV (sparse matrix vector product) $A \cdot b$: $\mathcal{O}(nnz)$

SpGEMM $A \cdot B$: $\mathcal{O}(nnz(A) + flops) \rightarrow$ Sparse Accumulator (SPA)



Kapitel 4

Vorlesung IV

4.1 direkte Lösungsverfahren

Voraussetzung: A invertierbar, d.h. $\det(A) \neq 0$. (Sonst keine exakte Lösung)

Bespiele:

- LR-Zerlegung (allg. Matrizen)
- Cholesky-Zerlegung (spd. Matrizen)
- QR-Zerlegung (allg. Matrizen)

Vorteil: exakte Lösung nach Faktorisierung schnell

Nachteil: kubische Laufzeit, dichtere Zwischenergebnisse

4.2 Cholesky-Zerlegung

Voraussetzung: $A \in \mathbb{R}^{n \times n}$ ist symmetrisch und positiv definit.

Zerlegung: $A = LL^T$ mit $L = (\ell_{ij}) \in \mathbb{R}^{n \times n}$ untere Dreiecksmatrix.

Verfahren:

$$\ell_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} \ell_{kj}^2}$$
$$\ell_{ik} = \frac{1}{\ell_{kk}} \left(a_{ik} - \sum_{j=1}^{k-1} \ell_{ij} \cdot \ell_{kj} \right)$$

Laufzeit: $\mathcal{O}(n^3) = \mathcal{O}(\#Multiplikation + \#Division + \#Wurzeln)$



4.3 Iterative Lösungsverfahren

Raten einer Anfangslösung x_0 und dann iterative Verbesserung dieser.

Mit M als Approximation von A^{-1} .

$$x^{(t+1)} = x^{(t)} - M (Ax^{(t)} - b)$$

Jacobi-Verfahren (Splitting-Verfahren)

Sei $A \in \mathbb{R}^{n \times n}$, $D = \text{diag}(a_{11}, \dots, a_{nn})$ und $Ax = b$ das zu lösende LGS. Dann gilt für die i -te ($i = 1, \dots, n$) Komponente der iterierten Lösung des nächsten Schrittes

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

Das Verfahren ist gut parallelisierbar, da es komponentenweise vorgeht und auf $x^{(t+1)}$ nur schreibend und $x^{(t)}$ lesend zugegriffen wird.



Kapitel 5

Vorlesung V

5.1 Konjugierte Gradienten (CG)

Voraussetzung: Matrix A ist symmetrisch und positiv definit.

Residuum: $r := b - Ax^{(t)}$

Verfahren:

func CG(A , b) \longrightarrow x :

Wähle $x_0 \in \mathbb{R}^{n \times n}$; $p_0 := r_0 := b - A \cdot x_0$; $\alpha_0 := \|r_0\|_2^2$ und $t := 0$

while: $\alpha_t \neq 0$

$$v_t := Ap_t$$

$$\lambda_t := \frac{\alpha_t}{(v_t, p_t)_2}$$

$$x_{t+1} := x_t + \lambda_t p_t$$

$$r_{t+1} := r_t - \lambda_t v_t$$

$$\alpha_{t+1} := \|r_{t+1}\|_2^2$$

$$p_{t+1} := r_{t+1} + \frac{\alpha_{t+1}}{\alpha_t} p_t$$

$$t := t + 1$$

end

Laufzeit: Nach n Iterationen exakte Lösung. typisch 2D $\mathcal{O}(n^{\frac{3}{2}})$ und 3D $\mathcal{O}(n^{\frac{4}{3}})$

\implies schneller als Jacobi-Verfahren



5.2 Parallelität

nötig, um ausreichend Speicher und/oder Rechengeschwindigkeit zu haben

hilfreich, weil heute alle Rechner Parallelrechner sind und um Energie zu sparen

Nebenläufigkeit: Verschiedene Operationen können gleichzeitig durchgeführt werden (Bsp. Zugriff auf RAM, arithmetische Operationen)

räumliche Parallelität:

- Komponenten im Rechner werden dupliziert
- Mehrere Komponenten können gleichzeitig gleichartige Operationen durchführen
- Beispiel: Addition

zeitliche Parallelität:

- Überlappung von synchronisierten Operationen mit ähnlicher Dauer
- Modell: Fließband (*pipelining*)

Kategorien von parallelen Systemen

Einzelrechner mit gemeinsamen Speicher:

- Mehrkern CPUs
- Multiprozessor-Knoten
- Kommunikation durch Zugriff auf gem. Speicher

Parallelrechner mit verteiltem Speicher:

- Mehrere Rechnerknoten, jeder mit eigenem (privaten) Speicher
- Verbunden durch ein Netzwerk
- Kommunikation durch Nachrichtenaustausch



5.3 Performance-Kriterien

Anwender wollen möglichst kurze Rechenzeit!!!

Grad der Parallelität eines Programms bezeichnet die Anzahl der Aufgaben, die gleichzeitig ausgeführt werden können.

gesamte Laufzeit = Laufzeit des sequentiellen Anteils + Laufzeit des parallelen Anteils

Beschleunigung und Effizienz

Speedup mit p Processunits (PUs) für Eingabe der Größe n :

$$S_p(n) = \frac{T_1(n)}{T_p(n)}$$

mit $T_1(n)$ die Laufzeit des schnellsten sequenziellen Algorithmus und $T_p(n)$ die Laufzeit des zu bewertenden Algorithmus (mit p PUs)

Effizienz: $\frac{S_p(n)}{p}$

Amdahls Gesetz

Annahme: Der parallele Anteil kann perfekt parallelisiert werden.

Dann gilt $T_p = T_{seq} + \frac{T_{par}}{p}$.

Amdahls Gesetz gibt eine obere Schranke für die Beschleunigung an bei Anteil α des parallelisierbaren Anteils:

$$S_p(n, \alpha) = \frac{1}{(1 - \alpha) + \frac{\alpha}{p}}$$

Beispiel: 10% parallelisierbar; Speedup höchstens 1,11 (bei $p \rightarrow \infty$)

Skalierbarkeit

starke Skalierbarkeit eines Algorithmus A bewertet die Effizienz von A bei steigender PU-Zahl, während die Problemgröße konstant bleibt.

schwache Skalierbarkeit eines Algorithmus A bewertet die Effizienz von A bei steigender PU-Zahl, während die Problemgröße in gleicher Weise steigt.



5.4 Parallele Basiskonzepte

Datenparallelität: Mehrere Datenblöcke werden gleichzeitig verarbeitet

(Bsp. Vektorsummen)

Eine **Schleife** heißt **parallelisierbar**, falls jede Permutation von Iterationen das selbe Resultat liefert.

Vektorisierung:

SIMD - single instruction, multiple data

Beispiel: ADDSUBPD - Add-Subtract-Packed-Double

Eingabe $\{A0, A1\}, \{B0, B1\} \longrightarrow$ Ausgabe $\{A0 - B0, A1 + B1\}$

Abhängigkeiten

- . I : Menge von Instruktionen
- . $\text{In}(I)$: Eingabedaten von I
- . $\text{Out}(I)$: Ausgabedaten von I

Datenabhängigkeit von I_A und I_B , falls: $\text{Out}(I_A) \cap \text{In}(I_B) \neq \emptyset$

Ausgabeabhängigkeit von I_A und I_B , falls: $\text{Out}(I_A) \cap \text{Out}(I_B) \neq \emptyset$

Eine **Schleife** ist **parallelisierbar** gdw. zwei Mengen von Instruktionen (die zu verschiedenen Iterationen gehören) keine Daten- und Ausgabeabhängigkeit haben.

Reduktion

Berechnung eines Einzelwertes aus einer Sequenz. In der Regel mit assoziativen Operationen, dadurch ist die Reihenfolge der Abarbeitung egal.

Beispiel: $+$, \cdot , AND, OR, XOR

ACHTUNG: Ausgabeabhängigkeit



Kapitel 6

Vorlesung VI

6.1 Parallele Systeme mit gemeinsamem Speicher

Einordnung:

- Mehrere PUs greifen auf gemeinsamen physischen Speicher zu
- Kommunikation über Lesen/Schreiben im gemeinsamen Speicher

Abgrenzung: virtueller gemeinsamer Speicher

Idee

mehrere Prozesse für Parallelität ist teuer → “Leichtgewichtige” Prozesse (Threads)

Beispiel: Multithreading mit OpenMP (`#pragma omp parallel`)

Vorteil:

- sequentieller Code lässt sich leicht erweitern
- Lohnt sich für kleinere (Alltags-)Probleme, da nicht alle ständig an ClusterRechnern arbeiten

6.2 Parallele Systeme mit verteiltem Speicher

Einordnung:

- Manche PEs/PUs können zwar gemeinsamen Speicher haben. Aber in der Regel: PUs haben privaten Speicher
- Kommunikation mit anderen PU-Einheiten per Nachricht (über (meist) schnelles Netzwerk)



- Rechenknoten i.d.R. “ähnlich” und physisch nah beieinander
- Eher hohe Ausfallsicherheit

Abgrenzung zu verteiltem System:

- Kaum Annahmen über Rechenknoten (können sehr unterschiedlich sein)
- Rechenknoten können geographisch stark verteilt sein (Netzwerk daher eher langsam)
- Hohe Ausfallraten
- Beispiel: IoT

Idee

Kommunikation zwischen Prozessen durch Nachrichtenaustausch

SPMD-Paradigma: single program, multiple data

Herausforderung: Effiziente algorithmische Umsetzung

6.3 MPI (Message Passing Interface)

Schnittstelle, die genormt („standardisiert“) wird – teilweise mit Empfehlungen/Anforderungen für eine Implementierung

Bsp. für Bibliothek: OpenMPI (Implementierung der Schnittstelle)

Vorteile

- Kein vendor lock-in mehr
- Code ist (prinzipiell) portabel
- Kommunikationsroutinen werden von Experten geschrieben (**Schichtenansatz**)
- Anwendungsprogrammierer muss Details der Hardware nicht kennen (kann aber helfen)
- Hohe Performance



Nachteile

- Aufwändiger Umbau von sequentielllem Code
- Debugging schwieriger als bei sequentielllem Code
- Erfordert recht viel Expertise

⇒ **Fazit:** Lohnt sich vor allem bei HPC-Anforderungen

Punkt-zu-Punkt-Kommunikation (P2P)

MPI_Send(

```
void* data, # initial address of send buffer  
int count, # number of elements in send buffer  
MPI_Datatype datatype, # datatype of each send buffer element  
int destination, # rank of destination  
int tag, # message tag (integer)  
MPI_Comm communicator # communicator  
)
```

MPI_Recv(

```
void* data, # initial address of receive buffer  
int count, # number of elements in recv buffer  
MPI_Datatype datatype, # datatype of each recv buffer element  
int source, # rank of source  
int tag, # message tag or MPI_ANY_TAG (int)  
MPI_Comm communicator, # communicator  
MPI_Status* status # status object (outgoing parameter)  
)
```

Hinweis: Diese Methoden warten immer, bis sie „fertig“ sind und setzen dann erst mit der Ausführung fort

