Zusammenfassung WR

Samuel Brinkmann (Matr. 624568)

8. Februar 2023

Inhaltsverzeichnis

Kapitel 1

Vorlesung I

1.1 Einführung

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

$$\text{Wissenschaftlicher Rechnen} \iff \text{Rechnergestützte (Ingenieur-))Wissenschaft}$$
$$\text{Entwurf von Verfahren} \iff \text{Verwendung dieser Verfahren}$$

Workflow:
physikalisches Problem
$\Longrightarrow$ mathematisches Modell
$\Longrightarrow$ numerisches Modell
$\Longrightarrow$ 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
$\longrightarrow$ Gleichungen geben nur Kopplung der Variablen an und keine tatsächlichen Werte
Lösung: Ableitungen werden durch den Differenzenquotienten approximiert

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

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 $x_N = 1$:

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

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)
$(\hat{A}_h)_{ij} = -\frac{1}{h^2}$, falls Knoten i randnah und j ein Nachbar mit 5-Punkte-Stern ist
$(\hat{A}_h)_{ij} = 0$, 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 = Gradmatrix − 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 $a^{\mathrm{T}} \cdot b$: $\mathcal{O}(n)$
Skalarprodukt $a \cdot b^{\mathrm{T}}$: $\mathcal{O}(n)$
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

Alterantiven: 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 $A \cdot B$: $\mathcal{O}(nnz(A) + flops) \longrightarrow$ 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: $A = LL^{\mathrm{T}}$ 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 Verbessung dieser.
Mit M als Approximation von A^{-1}.

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

Jacobi-Verfahren (Splitting-Verfahren)

Sei $A \in \mathbb{R}^{n\times n}$, $D = \operatorname{diag}(a_{11}, \ldots, a_{nn})$ und $Ax = b$ das zu lösende LGS. Dann gilt für die i-te $(i = 1, \ldots, n)$ 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: $\alpha_t \neq 0$

$$
\begin{aligned}
v_t &:= Ap_t \\
\lambda_t &:= \frac{\alpha_t}{(v_t, p_t)_2} \\
x_{t+1} &:= x_t + \lambda_t p_t \\
r_{t+1} &:= r_t - \lambda_t v_t \\
\alpha_{t+1} &:= \|r_{t+1}\|_2^2 \\
p_{t+1} &:= r_{t+1} + \frac{\alpha_{t+1}}{\alpha_t} p_t \\
t &:= t + 1
\end{aligned}
$$

end

Laufzeit: Nach n Iterationen exakte Lösung. typisch 2D $\mathcal{O}(n^{\frac{3}{2}})$ und 3D $\mathcal{O}(n^{\frac{4}{3}})$
$\Longrightarrow$ 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 gleichzeitg gleichartige Operationen durchführen
- Bespiel: 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 parallelisert 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 $p \longrightarrow \infty$)

Skalierbarkeit

starke Skalierbarkeit eines Algorithmus A bewertet die Effizienz von A bei steigender PU-Zahl, während die Problemgröße konstant beleibt.
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 Permutaion von Iterationen das selbe Resultat liefert.

Vektorisierung:

SIMD - single instruction, multiple data
Beispiel: ADDSUBPD - Add-Substract-Packed-Double
Eingabe {A0, A1}, {B0, B1} $\longrightarrow$ Ausgabe {A0 - B0, A1 + B1}

Abhängigkeiten

. I: Menge von Instruktionen

. In(I): Eingabedaten von I

. 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, dadruch ist die Reihenfolge der Abarbeitung egal.
Beispiel: +, ·, 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 $\longrightarrow$ "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 sequentiellem Code
- Debugging schwieriger als bei sequentiellem Code
- Erfordert recht viel Expertise

$\Longrightarrow$ **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

Kapitel 7

Vorlesung VII

7.1 Unterschiede OpenMP - MPI

OpenMP (Parallelität durch Multithreading)

- Zugriff auf gemeinsamen Speicher (Kommunikation implizit!)
- Beispiel: parallele Schleifen durch mehrere Threads

MPI (Parallelität durch verschiedene Prozesse)

- Jeder Prozess arbeitet auf seinem Teil der Daten
- Explizite Verteilung der Daten nötig
- Kommunikation durch entsprechende Routinen (explizit!)
- Programm wird parallel auf jedem Prozess abgearbeitet (SPMD, alles „gleichzeitig“)

7.2 Nicht-blockierende P2P-Kommunikation

Vorteil: Überlappung von Kommunikation und Berechnung oft sinnvoll
$\Longrightarrow$ Zeitgewinn
Beachte: Manchmal muss man warten, dass alle Eregbnisse eingetroffen sind (Programmkorrektheit)
Befehle: MPI_Isend und MPI_Irecv

7.3 Kollektive Kommunikation

Bedeutung: Kommunikationsoperationen, die alle Prozesse im Kommunikator einbeziehen

Vorteil:

- Einfacher zu programmieren
- Häufig auch deutlich schneller

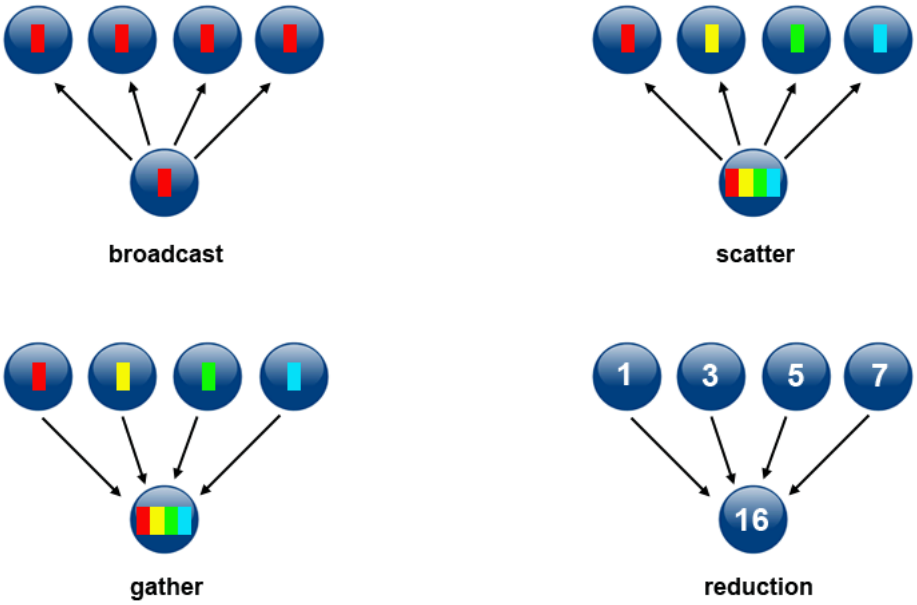
Arten:

- Synchronisation (alle warten an Sync.punkt: MPI_Barrier)
- Datenbewegung (einer an alle / von allen, alle an alle und von allen)
- Kollektive Berechnung (Reduktionen)

MPI_Barrier

- Barriere dient der Synchronisation
- Keiner darf weiter, solange nicht alle hier ankommen
- Verlangsamt das parallele Programm erheblich, aber manchmal einfach nötig

Kollektive Kommunikation



Befehle: MPI_Bcast, MPI_Scatter, MPI_Gather und MPI_Reduce
Hinweis: Alle kollektiven Operationen müssen von allen Prozessen im Kommunikator aufgerufen werden!

Kapitel 8

Vorlesung VIII

8.1 Wiederholung

Wie implementiert man LA Operationen parallel? (selbst Überlegen)
Wann ist eine horizontale und wann vertikale Aufteilung einer Matrix vorteilhaft?
vertikal: weniger Kommunikation für die Aufteilung der Daten von x bei $y = Ax$
horizontal: wenn man mit Teilen des Ergebnisvektors in dem gleichen Prozess weiterrechnen möchte. Bei $(y_1, y_2, y_3)^T = (A_1, A_2, A_3)^T \cdot (x_1, x_2, x_3)^T$ kann y_2 von Prozess 2 ermittelt werden mit $y_2 = A_2 \cdot (x_1, x_2, x_3)^T$ ohne eine Reduktion am Ende durchzuführen.

8.2 Parallel SpMV (Vorüberlegung)

Normalerweise sind die Matrizen *sparse* und die Vektoren *dense*
$\Longrightarrow$ Wir wollen möglichst viele Dateneinträge in der Matrix in dem Diagonalblock, um die Kommunikation bei einer horizontalen Aufteilung zu reduzieren.

Matrix Betrachtung

- **Diagonalblock:** interne Kommunikation
- **nicht-Diagonalblock:** benötigt Kommunikation (Minimierung Kommunikationsvolumen $\Longleftrightarrow$ Minimierung der nicht-Nulleinträge in nicht-Diagonalblöcken)

Graphen Betrachtung

- **interne Kanten** *(= zwischen Knoten gleicher Farbe)***:** interne Kommunikation

- **externe Kanten:** benötigt Kommunikation (Minimierung Kommunikationsvolumen $\Longleftrightarrow$ Minimierung der Anzahl der Kanten zwischen versch. Farben)

8.3 Parallel SpMV (Umsetzung)

Angenommen die Matrix ist gut partitioniert (Bspw. durch Permutaion der ursprünglichen Matrix).

Halo Knoten

jeder Prozess speichert neben den eigenen Knoten auch die Nachbarn seiner Grenzknoten (zwischen den Iterationen müssen die Halo Knoten ausgetauscht werden)

Parallel CSR Matrix mit Halo Support

Aufteilung der Matrix in quadratische Blöcke, welche als CSR Matrizen gespeichert werden (AUFPASSEN lokale und globale Indezierung beachten). Falls die Matrix sich nicht verändert, bleiben auch die Halo Knoten die gleichen bei neuen Vektoren. Da Matrix gut partitioniert ist, reicht oftmals eine P2P Kommunikation, da nur wenige nnz Werte in den nicht-Diagonalblöcken sind.

Halo Knoten Kommunikationsplan

Wie tauschen sich die Prozesse am besten aus? Bspw. kann 1-2 und 3-4 gleichzeitig ausgetauscht werden
$\Longrightarrow$ Kantenfärbungsproblem
Optimiere Laufzeit, in dem Kommunikationszeit parallel für Berechnungen genutzt wird (Bsp. MPI_Irecv und am Ende ein MPI_Wait nutzen)

Kapitel 9

Vorlesung IX

9.1 2D-Layout / -Verteilung

Problem

Graphen mit vielen Knoten mit eher kleinem Grad und wenigen mit einem sehr hohen Grad (Bsp. FaceBook)
Folge: Viele Teildomänen (Blöcke) haben viele andere Teildomänen als Nachbarn
Konsequenzen für 1D-Layouts:

- Partitionierung besonders wichtig, aber auch schwierig
- Viele Teilblöcke im Matrix-Streifen enthalten NNEs
- Quotientengraph hat viele recht hohe Knotengrade
- Jeder Prozess muss mit recht vielen anderen Prozessen Daten austauschen

Aufteilung 2D-Layout

$\sqrt{p}$ Blöcke der Höhe $\frac{n}{\sqrt{p}} \longrightarrow$ Redundante Speicherung: $\sqrt{p}$ Prozesse halten einen Block

Vorgehen für 2D-SpMV (siehe Foliensatz 10 Folie 16-20)

1. Austausch der relevanten Teile des Vektors x
2. Lokales Matrix-Vektor-Produkt berechnen
3. Akkumulation per Reduktion innerhalb des waagrechten Matrix-Streifens

9.2 Präkonditionierung

Konvergenz von zum Beispiel CG-Verfahren hängen von der Konditionszahl der Matrix A ab.
Def.: LGS $Ax = b$ wird ersetzt durch besser konditioniertes LGS $P_L A P_R x^P = P_L b$ mit $x = P_R x^P$.
Hierbei ist P_L bzw. P_R eine Annäherung an A^{-1}, da, falls $P = A^{-1}$, das LGS bereits gelöst wäre.
Skalierung: Eine reguläre Diagonalmatrix $D = diag\{d_{11}, \ldots, d_{nn}\} \in \mathbb{R}^{n \times n}$

Präkonditionierung mit Skalierung:

+ einfach
+ wenig Speicherplatz
+ schnell zu berechnen
- eher selten effektiv

Beispiel:

- Diagonalelemente $d_{ii} = \frac{1}{a_{ii}}$
- Betragsnorm $d_{ii} = \frac{1}{\sum_{j=1}^{n} |a_{ij}|}$
- Euklidische Norm $d_{ii} = \frac{1}{\left(\sum_{j=1}^{n} |a_{ij}|^2\right)^{1/2}}$
- Maximumsnorm $d_{ii} = \frac{1}{\max_{j=1,\ldots,n} |a_{ij}|}$

Alternativ: Polynomiale Präkonditionierung $p^{(m)} := \sum_{k=0}^{m} (I - A)^k$

Symmetrie

Falls A spd. ist, muss auch P spd. sein, aber PA muss nicht länger symmetrisch sein!!!
Folge: CG-Verfahren ist nicht ohne Weiteres auf $PAx = Pb$ anwendbar
Lösung: Da P spd., gilt $P = L^{\mathrm{T}} L$ und somit $L^{\mathrm{T}} A L \hat{x} = L^{\mathrm{T}} b$ mit $x = L\hat{x}$

Präkonditioniertes CG-Verfahren (PCG)

Verwendung der obigen symmetrieerhaltenden Präkonditionierung und einige Anpassung beim CG-Verfahren, wobei die Operationen im Wesentlichen unverändert sind. (Genaueres siehe Foliensatz 10 Folie 30-31, könnte Klausur relevant sein, schien mir aber ein bisschen komplex dafür)

Kapitel 10

Vorlesung X

10.1 PageRank aus mathematischer Sicht

Modell des Zufallssurfers

- Webgraph: Webseite ist Knoten, Link ist gerichtete Kante
- Surfer bewegt sich zufällig im Webgraphen
- Klick auf Link: Ausgehender Kante wird mit gleicher Wkt. gefolgt
- Lesezeichen / neue URL eintippen: “Wegteleportieren”
- Stochastischer Prozess, stationärer Zustand ist PageRank-Vektor

Mathematische Modellierung

- Dämpfungsfaktor α: Mit Wkt. α Link klicken, mit Wkt. 1 - α teleportieren
- Linkverfolgung: Transitionsmatrix P

$$P_{ij} = \text{Wkt. auf Seite } i \text{ auf Link für Seite } j \text{ zu klicken}$$

- Teleport-Vektor y, stochastisch, $\|y\|_1 = 1$
- Google-Matrix $G = \alpha P + (1 - \alpha)\mathbf{1}y^T$

10.2 Potenzmethode

Basis-Verfahren

so nur für dominanten Eigenvektor

$$x^{(k+1)} = cAx^{(k)}$$

mit Normalisierungskonstante c, dass $x^{(k+1)}$ nicht zu groß wird.
$x^{(k+1)}$ konvergiert für k "groß genug" gegen Eigenvektor v_1 von A, wobei $\lambda_1 = \frac{\|x^{k+1}\|}{\|x^k\|}$ der dominante Eigenvektor ist.
Konvergenz bestimmt durch Verhältnis $\frac{|\lambda_2|}{|\lambda_1|}$, wobei je kleiner das Verhältnis, desto bessere Konvergenz ($\lambda_1 > \lambda_2 \geq \lambda_i$ mit $i \geq 3$)

Zusammenhang Zufallssurfer

$$x^T_{(t+1)} := x^T_{(t)} G$$

mit $P = D^{-1}A$ und x Vektor der Wahrscheinlichkeitsmasse (stationär: PageRank-Vektor)

Erklärung Konvergenz

$$x^{(0)} = c_1 v_1 + c_2 v_2 + \cdots + c_n v_n \qquad \text{Startvektor in Eigenvektoren-Basis}$$

$$x^{(k)} = Ax^{(k-1)} = \cdots = A^k x^{(0)} = \sum_{i=1}^{n} c_i \lambda_i^k v_i$$

$$\frac{x^{(k)}}{c_1 \lambda_1^k} = v_1 + \frac{c_2}{c_1}\left(\frac{\lambda_2}{\lambda_1}\right)^k v_2 + \cdots + \frac{c_n}{c_1}\left(\frac{\lambda_n}{\lambda_1}\right)^k v_n$$

Da λ_1 dominant: alle Terme rechts von v_1 gehen für große k gegen 0.