Zusammenfassung WR

Samuel Brinkmann (Matr. 624568)

15. Februar 2023

Inhaltsverzeichnis

Inhaltsverzeichnis **2**

1 Vorlesung I **5**
1.1 Einführung 5
1.2 Modellierung 6
1.3 mathematisches Modell zu numerischem Modell 6

2 Vorlesung II **7**

3 Vorlesung III **8**
3.1 Finite Differenzen in 2D 8
3.2 LinA Wiederholung 9

4 Vorlesung IV **10**
4.1 Speicher 10
4.2 Dichte Vektor-Matrix-Operationen 10
4.3 Dünn besetzte Matrizen 11

5 Vorlesung V **12**
5.1 direkte Lösungsverfahren 12
5.2 Cholesky-Zerlegung 12
5.3 Iterative Lösungsverfahren 13

6 Vorlesung VI **14**
6.1 Konjugierte Gradienten (CG) 14
6.2 Parallelität 15
6.3 Performance-Kriterien 16
6.4 Parallele Basiskonzepte 17

7 Vorlesung VII **18**
7.1 Parallele Systeme mit gemeinsamem Speicher 18
7.2 Parallele Systeme mit verteiltem Speicher 18
7.3 MPI (Message Passing Interface) 19

8 Vorlesung VIII **21**
8.1 Unterschiede OpenMP - MPI 21
8.2 Nicht-blockierende P2P-Kommunikation 21

8.3 Kollektive Kommunikation 22

9 Vorlesung IX **24**
9.1 Wiederholung 24
9.2 Parallel SpMV (Vorüberlegung) 24
9.3 Parallel SpMV (Umsetzung) 25

10 Vorlesung X **26**
10.1 2D-Layout / -Verteilung 26
10.2 Präkonditionierung 27

11 Vorlesung XI **29**
11.1 PageRank aus mathematischer Sicht 29
11.2 Potenzmethode 30
11.3 Satz von Perron-Frobenius 31
11.4 Eigenlöser im Überblick 31

12 Vorlesung XII **33**
12.1 Lanczos-Verfahren 33
12.2 Graphpartitionierung 35
12.3 Spektrale Partitionierung 35

13 Vorlesung XIII **38**
13.1 Mehrebenen-Verfahren (Graphpartitionierung) 38
13.2 Vergröberung (Mehrebenen-Verfahren) 40
13.3 Approximation Matchings 41

14 Vorlesung XIV **43**
14.1 Label Propagation als lokale Suche 43
14.2 Label Propagation zur Vergröberung 44
14.3 Label Propagation parallel 45
14.4 Hinführung zum Mapping-Problem 45

15 Fragen aus den Folien **47**
15.1 Vorlesung I 47
15.2 Vorlesung II 47
15.3 Vorlesung III 48
15.4 Vorlesung IV 49
15.5 Vorlesung V 49

15.6 Vorlesung VI 50
15.7 Vorlesung VII 51
15.8 Vorlesung VIII 51
15.9 Vorlesung IX 52
15.10 Vorlesung X 52
15.11 Vorlesung XI 53
15.12 Vorlesung XII 53
15.13 Vorlesung XIII 54
15.14 Vorlesung XIV 54

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

Partielle Ableitungen, PDEs, Gradientenvektor, Finite Differenzen Einleitung
(Nur Grundlagen, siehe Foliensatz zur Vorlesung 2)

Kapitel 3

Vorlesung III

3.1 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
$f_{ij} = \frac{1}{h^2}(-u_{i,j-1} - u_{i-1,j} + 4u_{i,j} - u_{i+1,j} - u_{i,j+1})$

3.2 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 4

Vorlesung IV

4.1 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).

4.2 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)

4.3 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 5

Vorlesung V

5.1 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

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

5.3 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 6

Vorlesung VI

6.1 Konjugierte Gradienten (CG)

Voraussetzung: Matrix A ist symmetrisch und positiv definit.
Residuum: $r := b - Ax^{(t)}$
Verfahren:

func CG(A, b) $\longrightarrow$ x:
Wähle $x_0 \in \mathbb{R}^{n\times n}$; $d_0 := r_0 := b - A \cdot x_0$; $\alpha_0 := \|r_0\|_2^2$ und $t := 0$
while: $\alpha_t \neq 0$

$$
\begin{aligned}
q &:= Ad_t \\
\lambda_t &:= \frac{\alpha_t}{(d_t, q)_2} \\
x_{t+1} &:= x_t + \lambda_t d_t \\
r_{t+1} &:= r_t - \lambda_t q \\
\alpha_{t+1} &:= \|r_{t+1}\|_2^2 \\
d_{t+1} &:= r_{t+1} + \frac{\alpha_{t+1}}{\alpha_t} d_t \\
t &:= t + 1
\end{aligned}
$$

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

6.2 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

6.3 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.

6.4 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 7

Vorlesung VII

7.1 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

7.2 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

7.3 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 8

Vorlesung VIII

8.1 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“)

8.2 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

8.3 Kollektive Kommunikation

Bedeutung: Kommunikationsoperationen, die alle Prozesse im Kommunikator einbeziehen

Vorteil:

- Einfacher zu programmieren
- Häufig auch deutlich schneller

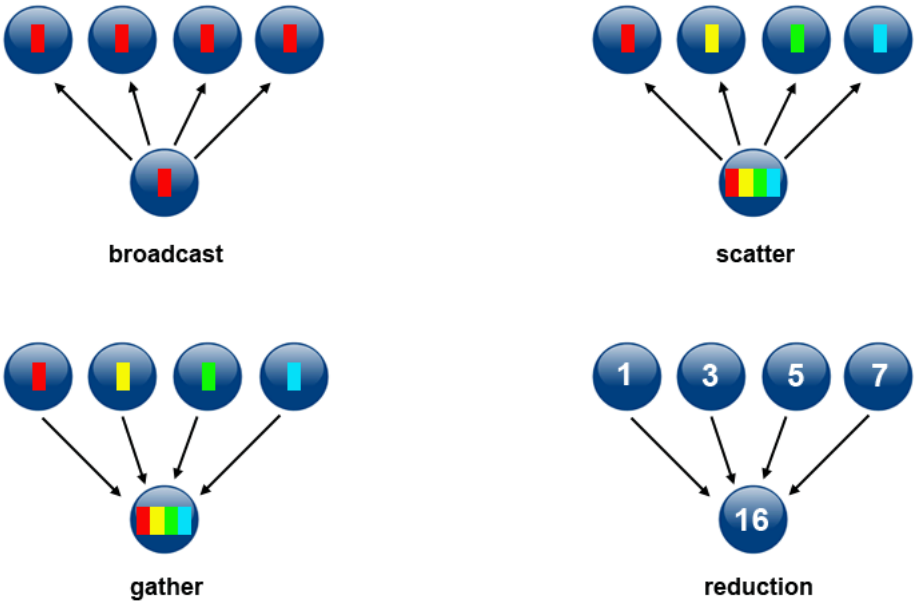
Arten:

- 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 9

Vorlesung IX

9.1 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.

9.2 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)

9.3 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 10

Vorlesung X

10.1 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

10.2 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 11

Vorlesung XI

11.1 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 $G = \alpha P + (1 - \alpha)\mathbf{1}y^T$

11.2 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.

Diagonalverschiebung

Konvergenz kann verbessert werden durch Verschiebung der EW.
So eine Verschiebung σ wird erreicht durch $(A - \sigma I)$ statt A. Neue Konvergenzrate:

$$\left|\frac{\lambda_2 - \sigma}{\lambda_1 - \sigma}\right| \qquad \text{(Beste Verschiebung: } \sigma = 1/2(\lambda_2 + \lambda_n) \text{ oder } \sigma = 1/2(\lambda_1 + \lambda_{n-1})\text{)}$$

Inverse Potenzmethode

Wenn A regulär und symmetrisch: EW der Inversen A^{-1} sind invers
Also statt A mit A^{-1} iterieren: $Ax^{(k+1)} = cx^{(k)}$
(auch hier kann eine Diagonalverschiebung vorgenommen werden)
Konvergenzrate:

$$\left| \frac{\lambda_n - \sigma}{\lambda_{n-1} - \sigma} \right|$$

11.3 Satz von Perron-Frobenius

Sei A eine positive Matrix.

- Dann ist der größte Eigenwert λ_1 von A positiv und einfach (nur einmal vorkommend)
- Alle anderen Eigenwerte sind demnach betragsmäßig kleiner als λ_1
- Außerdem existiert zu λ_1 ein ebenfalls positiver Eigenvektor x

11.4 Eigenlöser im Überblick

Householder-Transformation

Voraussetzung: dominanter Eigenwert und Eigenvektor sind bekannt
Nutzung: Suchen von zwei bis drei subdominante Eigenwerte
Vorgehen: rekursiv, welches in jedem rekursiven Aufruf die Matrix um eine Zeile und Spalte verkleinert

QR-Verfahren

Voraussetzung: A symmetrisch
Nutzung: Bestimmung der reellen Eigenwerte (somit aller, da A symm.)
Vorgehen: Sequenz von *ähnlichen* Matrizen konstruieren:

$$A_1, A_2, \ldots, A_k,$$

wobei A_k für große k gegen eine obere Dreiecksmatrix konvergiert (falls alle Eigenwerte betragsmäßig unterschiedlich sind), deren Diagonaleinträge die reellen Eigenwerte von A sind.

Lanczos

Voraussetzung: keine, aber besonders geeignet für große symmetrische und dünn besetzte Matrizen
Nutzung: Bestimmung von $j = 1, \ldots, n$ Eigenwerte
Vorgehen: Iteratives Verfahren, welche die vorherigen Iterierten $v, Av, A^2v, \ldots, A^{k-1}v$ nutzt. Diese werden nach der Art der Potenzmethode erzeugt. Die k Vektoren bilden einen Krylov-Unterraum
Vorteil gegenüber QR-Verfahren: Lanczos kann nach j Schritten beendet werden für die ersten j Eigenwerte/-vektoren und im Gegensatz zum QR-Verfahren erhält es den Füllstand der iterierten Matrizen (effizienter)

Kapitel 12

Vorlesung XII

12.1 Lanczos-Verfahren

Kernidee

Iteratives Verfahren, welche die vorherigen Iterierten $v, Av, A^2v, \dots, A^{k-1}v$ nutzt. Diese werden nach der Art der Potenzmethode erzeugt. Wir definieren $\{q_1, q_2, \dots, q_k\} := \{v, Av, \dots, A^{k-1}v\}$ und $Q_k = (q_1 q_2 \cdots q_k)$. Dann ist

$$T_k := Q_k^T A Q_k = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \cdots & & \mathbf{0} \\ \beta_1 & \alpha_2 & \beta_2 & \cdots & & \\ 0 & \beta_2 & \ddots & \ddots & & \\ \vdots & \vdots & \ddots & & & \\ & & & & & \beta_{k-1} \\ \mathbf{0} & & & & \beta_{k-1} & \alpha_k \end{bmatrix}$$

mit $\alpha_i = q_i^T A q_i$ und $\beta_i = \|r_i\|$ (folgt aus Rekurrenzgleichung).
Da gilt $Q^T A = TQ$, sind die Eigenwerte von T gleich denen von A (können aus T mit QR-Methode mit linearem Aufwand berechnet werden).

Rekurrenzgleichung

Tridiagonal Struktur von T ist Folge einer Rekurrenz mit drei Termen:

$$Aq_i = \beta_{i-1} 1_{i-1} + \alpha_i q_i + \beta_i q_{i+1} \qquad (i = 1, 2, \dots)$$

Anmerkungen

- $r_i := \beta_i q_{i+1} = Aq_i - \alpha_i q_i - \beta_{i-1} q_{i-1}$
- $q_{i+1} = \frac{r_i}{\beta_i}$ (Annahme $\beta_i \neq 0$)
- Alle Vektoren q_{i+1} stehen senkrecht auf q_k mit $k < i$ (folgt aus Symmetrie von A).
- Pro Iteration ein Matrix-Vektor-Produkt, sonst nur einfachere Basis-Operationen
- Speichereffizient (höchstens drei Vektoren nötig: r, q_j, q_{j-1})

Algorithmus

Wähle $q_0 = 0$; beliebigen Vektor $r \neq 0$ und $\beta_0 := \|r\|_2$
for $j := 1$ to n do

$$q_j = \frac{r}{\beta_{j-1}}$$
$$r = Aq_j$$
$$\alpha_j = q_j^T r$$
$$r = r - q_{j-1}\beta_{j-1}$$
$$r = r - q_j \alpha_j$$
Orthogonalisieren, falls notwendig
$$\beta_j = \|r\|_2$$

end

12.2 Graphpartitionierung

Graphpartitionierungsproblem

Sei Graph $G = (V, E)$ gegeben. Dann ist eine k-Partitionierung $\Pi = (V_1, \ldots, V_k)$ von V gesucht, so dass

- $|V_i| \leq \left\lceil \frac{|V|}{k} \right\rceil \cdot (1 + \epsilon)$ (Nebenbedingung: Balance)
- $\min\{e = \{u, v\} : \Pi(u) \neq \Pi(v)\}$ (Optimierungskriterium: Kantenschnitt)

Je nach Modell:

- Knoten zählen einmal, nicht pro Kante (Unterschied zur Zahl der Randknoten)
- Knoten zählen einmal pro benachbarten Block (Unterschied zum Kommunikationsvolumen) *(genauer)*
- Wir warten auf die langsamste PU $\longrightarrow$ Maximales Komm.volumen daher bessere Zielfunktion als Summe (Summe vs Maximum)

Kantenschnitt

Annahme: Suchen 2-Partition (V_1, V_2). Kodierung der Partition mit Indikatorvektor x

$$x(i) = \begin{cases} -1, i \in V_1 \\ +1, i \in V_2 \end{cases}$$

induzierter Kantenschnitt: $\frac{1}{4} x^T L x$ mit L Laplace-Matrix von G

12.3 Spektrale Partitionierung

GPP als BQP (binäres quadratisches Programm)

2-Partitionierung, Annahme: n gerade, $|V_1| = |V_2|$ balanciert

$$\min_x x^T L x \qquad \text{(Kantenschnitt als Opt.Kriterium)}$$

unter der Nebenbedingung $x^T \mathbf{1} = 0$ (perfekte Balance) und $x(i) \in \{-1, +1\}$
Beobachtung: $x^T x = n$
Theorem: Das Entscheidungsproblem zu GPP ist NP-vollständig.

Courant-Fischer-Theorem

Ist $A \in \mathbb{R}^{n \times n}$ symmetrisch mit aufsteigenden Eigenwerten $\lambda_1 \leq \cdots \leq \lambda_n$ und X_i die Menge der i-dimensionalen Untervektorräume von A. Dann

$$\lambda_i = \min_{x \in X_i} \max_{x \in X, x \neq 0} \frac{x^T A x}{x^T x}.$$

Folgerung für Laplace-Matrix: $\lambda_1 = 0$ und $\lambda_2 \perp \mathbf{1}$

Relaxierung des BQP

$$\min_{x \in \mathbb{R}^{n \times n}, x \neq 0} x^T L x \text{ und Balance } x^T \cdot \mathbf{1} = 0$$

Gesucht ist der EV zum EW λ_2.
Zusammenfassung der Herleitung des Algorithmus:

- Kodiere Partition mit Vektor und Graphen mit Matrix (Arithmetisierung des Problems)
- Formuliere Zielfunktion ($x^T L x$) und Nebenbedingungen ($x \neq 0, x^T \cdot \mathbf{1} = 0$)
- Relaxiere diskretes Problem zu kontinuierlichem
- Löse kontinuierliches Problem mit Eigenlöser
- Diskretisiere kontinuierliche Lösung

Spektrale Partitionierung

Eingabe: $G = (V, E)$
Ausgabe: 2-Partitionierung (V_1, V_2) von G
Verfahren: $V_1 := \varnothing$ und $V_2 := \varnothing$

1. Konstruiere L zu G
2. Berechne den Eigenvektor z_2 zum Eigenwert λ_2
3. Berechne den Median μ in z_2
4. Teile die Indizes i von z_2 auf: V_1 falls $z[i] \leq \mu$, ansonsten V_2
5. **return** (V_1, V_2)

Vorteile

- Schnell programmiert (Eigenlöser-Bibliothek vorausgesetzt)
- Globale Sicht, Herleitung erklärt vernünftige Qualität
- Einer der beiden Blöcke ist garantiert zusammenhängend
  (wenn G zusammenhängend ist)

Nachteile

c $\Longrightarrow$ **Fazit:** geeignet für schnellen Prototypen

Kapitel 13

Vorlesung XIII

13.1 Mehrebenen-Verfahren (Graphpartitionierung)

Graphpartitionierungsproblem wie oben definiert

Lokale Suche

Gegeben: (zulässige) Lösung für Optimierungsproblem
Idee: (Kleine) Veränderung hin zu einer anderen (ähnlichen) zulässigen Lösung durch sog. Nachbarschaftsoperator (iterativ).
Beispiel: Verändere die Blockzugehörigkeit eines Knotens, Vertausche die Blockzugehörigkeit von zwei Knoten
Anmerkungen:

- Größe der Nachbarschaft bestimmt Laufzeit und Qualität wesentlich
- Oft gilt: mit vernünftiger Startlösung kommt man zu guten Ergebnissen
- Oft nicht effizient: zig Startlösungen durchprobieren

Mehrebenen-Verfahren

Sicht der Lokalen Suche wird erweitert und dadurch bessere Lösungen

3 Phasen

1. Rekursive Vergröberung
2. Initiale Lösung
3. Wechsel von Lösungsinterpolation und lokaler Verbesserung der Interpolation

Rekursive Vergröberung

- Erzeugt Hierarchie von Graphen
- Jede gröbere Hierarchieebene hat weniger Knoten und Kanten
- Trotz Vergröberung bleibt Struktur der Eingabe (einigermaßen) erhalten

Initiale Lösung

- Spektrale Partitionierung
- Region growing
- Metaheuristiken

Interpolation

- Superknoten des KK gibt Blocknummer (Farbe) an alle Knoten im KK (= Kontraktionskern) weiter

Lokale Verbesserung

- Lokale Suche (viele Varianten)
- Flussbasiert
- Diffusionsbasiert

Grafik zum Verfahren

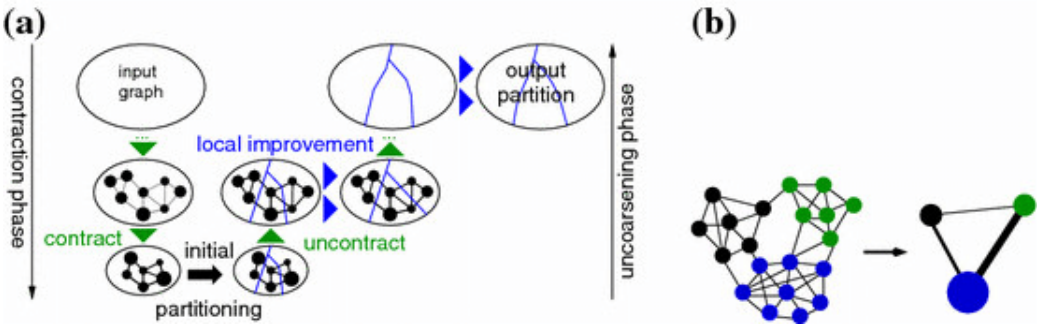


Abbildung 1: (a) Mehrebenen-Verfahren, (b) Kontraktionskerne

13.2 Vergröberung (Mehrebenen-Verfahren)

Allgemein: Vergröberung soll "Struktur" der Instanz möglichst gut erhalten
$\Longrightarrow$ Knoten zusammenfassen, die vermutlich in einen Block gehören
$\Longrightarrow$ Schwere Kanten möglichst nicht schneiden (gewichteter Kantenschnitt!)

Kontraktionskerne (KK)

Kontraktionskrene zu Superknoten: Kanten zwischen KKs verschmelzen, wobei sich Gewichte addieren. Bei GP: Kanten innerhalb KKs irgnorieren
$\Longrightarrow$ diese Art der Vergröberung erhält den Kantenschnitt!

Vergröberung mit Matchings

Sei $G = (V, E)$
Ein **Matching** M ist eine Teilmenge von E, bei der keine zwei Kanten einen Knoten gemeinsam haben. *(Anders ausgedrückt: Teilgraph mit Grad höchstens 1.)*
Nicht erweiterbares Matching (engl.: maximal matching): keine Kante kann hinzugefügt werden, ohne Matching-Eigenschaft zu verletzen.
Größtmögliches Matching (engl.: maximum matching): $|M|$ maximal unter allen Matchings von G.
M ist **Matching maximalen Gewichts**:

$$\sum_{e \in M} \omega(e) \text{ maximal unter allen Matchings in } G$$

KK Größe: Größe 2 (Matching-Kante) oder Größe 1 (ungematchter Knoten)
Alternative: größere KKs *(Cluster)*, welche intern dicht (viele Kanten) und nach außen dünn (wenige Kanten) sind

Vor- und Nachteile Größe der Kontraktionskerne

Klein (bspw. Matchings):

- Hierarchie wird tiefer
- Tiefere Hierarchie benötigt oft mehr Zeit (und Speicher), liefert aber oft bessere Qualität
- **Aber:** Matching-basiert funktioniert nicht gut bei vielen Stern-artigen Knoten

Groß (bspw. Cluster):

- Hierarchie wird flacher
- Flachere Hierarchie umgekehrt
- Gut bei Stern-artigen Knoten
- **Achtung:** Cluster-basiert funktioniert nur gut, wenn Graph genügend unterschiedliche Auflösungen hat

Gewichtete Matchings

Bei Vergröberung vermeiden: Schneiden von schweren Kanten
$\Longrightarrow$ Schwere Kanten möglichst in Kontraktionskerne packen
Daher: Matching mit maximalem Gewicht berechnen
Aber: Exakter Algorithmus braucht $\mathcal{O}(mn \log(n)$ bzw. $\mathcal{O}(mn + n^2 \log(n)$ Zeit

13.3 Approximation Matchings

α-Approximationsalgorithmus

Sei Π ein Optimierungsproblem.
Ein Algorithmus A heißt α-Approximationsalgorithmus, wenn A zu jeder Instanz I von Π eine zulässige *(aber nicht notwenidgerweise optimale)* Lösung mit $R_A(I) \leq \alpha$ berechnet, wobei

- $R_A(I) := A(I)/OPT(I)$, falls Π Minimierungsproblem
- $R_A(I) := OPT(I)/A(I)$, falls Π Maximierungsproblem

mit $A(I)$ Wert der Lösung, die A für I berechnet und $OPT(I)$ Wert einer optimalen Lösung für I.

PathGrowing-Algorithmus (Local Max)

func PathGrowingMWM($G = (V, E, w)$):
  $M_1 = \varnothing$, $M_2 = \varnothing$ und $i = 1$
  while: $E \neq \varnothing$ do
    Wähle $x \in V$ mit $\deg(x) > 0$ beliebig
    while: x hat Nachbarn in G do
      Sei $\{x, y\}$ die schwerste zu x inzidente Kante
      Füge $\{x, y\}$ zu M_i hinzu
      Entferne x aus G
      $i = 3 - i$
      $x = y$
    end
  end
  return $\text{argmax}(w(M_1), w(M_2))$

Laufzeit $\leq$ Summe der Kontengrade in V: $\mathcal{O}(|E|)$

Approximationsgüte

Sie M^* ein ideales Matching von G.
$\Longrightarrow$ Für jeden Vaterknoten einer Kante in M^* gibt es eine mind. so schwere Kante in M_1 oder in M_2
$\Longrightarrow w(M_1 \cup M_2) \geq w(M^*)$, da M_1 und M_2 disjunkt sind
$\Longrightarrow \max\{w(M_1), w(M_2)\} \geq \frac{1}{2} w(M_1 \cup M_2) \geq \frac{1}{2} w(M^*)$
$\Longrightarrow \alpha = 2$

Kapitel 14

Vorlesung XIV

14.1 Label Propagation als lokale Suche

Herleitung

Zielfunktion: Kantenschnitt
Interne Nachbarn: Nachbarn im eigenen Block
Externe Nachbarn: Nachbarn im anderen Block
Nachbarschaftsoperator: ein Knoten wechselt seine "Farbe"
Feststellung: Eine Partition ist nicht optimal, wenn ein Knoten mehr externe als interne Nachbarn hat (zu einem einzelnen externen Block) – und ein Wechsel die Balance nicht verletzt

Label Propagation

Idee: Knoten ordnet sich dem dominanten Block in der Nachbarschaft zu
Aber: Balance beachten! D.h., Knoten ordnet sich dem dominanten Block in der Nachbarschaft zu, der noch Knoten aufnehmen kann

Grundlegende Iteration

for phase $1, 2, \ldots$ do
  for jeden Knoten $v \in V$ do
    wende Nachbarschaftoperator auf v an *(mit Beachtung der Balance)*
  end
end

Abbruchkriterium

- Wenn alle Knoten in einer Phase „zufrieden“ sind (keine Verbesserung möglich, lokales Optimum)
- Nach einer festen Anzahl von Phasen (Rechenzeit)
- Wenn sich nur noch wenig verbessert (Rechenzeit)

14.2 Label Propagation zur Vergröberung

Clusterbasierte Vergröberung

Cluster: Knotenverteilung: intern dicht, extern dünn
Idee: Label Propagation mit Größenbeschränkung (Parameter für Clustergröße)
Initial: jeder Knoten ist sein eigener Cluster
Grundlegende Iteration: wie bei Label Propagation für lokale Suche, aber mit Parameter statt Balancekriterium zur Größenbeschränkung

Zusammenfassung

- Vergröberung:
  KKs bestimmen (z.Bsp. Matchings, Cluster), verschmelzen
- Initiale Lösung:
  Region growing, spektral, Label propagation (angepasst)
- Lokale Suche:
  Label propagation, andere Knotentauschheuristiken wie KL/FM
- **Wichtig:** Techniken müssen zueinander und zur Eingabe passen!

14.3 Label Propagation parallel

Ziel: Mit LP parallel partitionieren
Beobachtung: LP sehr ähnlich zu SpMV

Vergleich innere Schleifen

LP

Vorbehalt: Größenbeschränkung
for jeden Knoten $v \in V$ do
$l'(v) = \text{argmax multiplicity}\{l(u) : u \in N(v)\}$

SpMV

$A \cdot l = l'$
for jeden Knoten $v \in V$ do
$l'(v) = \sum_{u \in N(v)} a_{ij} l(v)$

1 statt 2 Vektoren

for jeden Knoten $v \in V$ do
$l(v) = \text{argmax multiplicity}\{l(u) : u \in N(v)\}$
Vorteil: schnellere Konvergenz *(Vergleich: Jacobi-Verfahren $\longrightarrow$ Gauß-Seidel-Verfahren)*
Nachteil: kompliziertere Umsetzung

14.4 Hinführung zum Mapping-Problem

Kommunikationskosten im NUMA-Fall

NUMA: non-uniform memory access
bisherige Annahme: jeder Prozess(or) kann mit jedem anderen Prozess(or) gleich schnell "reden" $\longrightarrow$ entspricht häufig nicht der Realität
$\Longrightarrow$ Unterschiedliche Kommunikationskosten mit einbeziehen!
(Gewichtung der Kanten zwischen Blöcken)

Anpassen der Problemformulierung

Gegeben

- Applikationsgraph $G_\alpha = (V_\alpha, E_\alpha)$ *(korrespondiert zu unserer Matrix)*
- Prozessorgraph $G_p = (V_p, E_p)$ *(modelliert paralleles System)*
- Zahl der Blöcke k *(entspricht meist $|V_p|$)*

Gesucht

- Partition Π von V_α in k (fast) gleich große Blöcke
- Abbildung der Blöcke auf konkrete Prozessoren bei der Π den um die Kommunikationskosten erweiterten Kantenschnitt minimiert

Kapitel 15

Fragen aus den Folien

15.1 Vorlesung I

- Was ist wissenschaftliches Rechnen?
- Welche Art von Problemstellungen werden mit Methoden des wissenschaftlichen Rechnens gelöst?
- Woraus besteht der übliche Workflow?
- Was ist der Unterschied zu computational science (and engineering)?
- Was ist ein Modell? Welche Phasen gehören zur Modellbildung?
- Wie kommen wir von der Differentialgleichung zu algebraischen Gleichungen?
- Welche Rolle spielt der Differenzenquotient dabei?

15.2 Vorlesung II

- Wie ist der Differenzenquotient einer partiellen Ableitung definiert?
- Leiten Sie konkrete Funktionen partiell ab!
- Wie ist der Gradientenvektor definiert?
- Was ist der Unterschied zwischen Vorwärts- und RückwärtsDiskretisierung (bildlich, in Formeln)?
- Erläutern Sie das Konzept der finiten Differenzen!

- Was charakterisiert Anfangswert-/Randwert-/Anfangs-Randwert-Aufgaben?
- Geben Sie die FD-Schemata für die Wärmeleitungsgleichung an!
- Welcher Zusammenhang besteht zwischen der Rückwärtsmethode und einem LGS?
- Zeigen Sie, dass dieses LGS eine Tridiagonalmatrix [G11, S. 57]) hat!
- Was kann beim Diskretisieren schiefgehen (und warum)?

15.3 Vorlesung III

- Wie sieht die FD-Approximation der Wärmeleitung in 2D aus (bildlich, als Formel)?
- Welche LGS-Matrix liefert ein 5-Punkt-Stern? Stellen Sie die Matrix für ein Gebiet der Größe $m \times n$ auf!
- Geben Sie die Definitionen der Vektor- und Matrix-Eigenschaften wieder
- Geben Sie Beispiele für bestimmte Eigenschaften an. Prüfen Sie an Beispielen nach, ob bestimmte Eigenschaften erfüllt sind oder nicht.
- Rechnen Sie mit den typischen Operationen (Skalarprodukt, Matrix-VektorMultiplikation, Matrix-Matrix-Multiplikation, ...) auf Beispielen.
- Leiten Sie die quadratische Form der Laplace-Matrix her.
- Welche geometrische Bedeutung haben LGS?
- Wie steht es um die Lösbarkeit von LGS?
- Wie ist die Laplace-Matrix eines Graphen definiert?
- Welche geometrischen Eigenschaften hat eine orthogonale Transformation (Längen, Winkel)?
- Was ist eine Orthonormalbasis?

15.4 Vorlesung IV

- Warum gibt es die Speicherhierarchie? Welche Grundprinzipien hat sie?
- Durch welche Prinzipien erreicht man eine gute Cache-Nutzung?
- Erläutern Sie räumliche und zeitliche Lokalität! Geben Sie Beispiele an!
- Warum ist die "richtige" Reihenfolge bei Schleifennestern wichtig?
- Wie lässt sich die Schulmethode zur Matrix-Multiplikation so anpassen, dass sie Cache-effizienter wird?
- Schreiben Sie die grundlegenden Matrix- und Vektoroperationen in Pseudocode auf!
- Wie funktioniert das CSR- bzw. CSC-Format? Worin unterscheiden sie sich? Erläutern sie die Datenstruktur am konkreten Beispiel!
- Welche Abwandlungen können Sie nennen? Zu welchem Zweck?
- Wie funktionieren die LA-Basisoperationen mit dem CSR-Format (Skizze/Beispiel sowie Pseudocode)?
- Was ist ein SPA (sparse accumulator)? Wozu wird er verwendet? Woraus besteht er? Wie funktioniert er? Welche Laufzeiten ergeben sich?

15.5 Vorlesung V

- Was ist ein SPA (sparse accumulator)? Wozu wird er verwendet? Woraus besteht er? Wie sieht das Interface aus? Wie funktioniert er? Welche Laufzeiten ergeben sich und warum?
- Wie funktioniert SpGEMM mit einem SPA und einer CSR- oder CSC-Matrix? Welche Laufzeiten ergeben sich und warum?
- Berechnen Sie den Lösungsvektor zu einer rechten Seite am Beispiel der gegebenen 4x4-Matrix!
- Führen Sie eine komplette Cholesky-Zerlegung anhand einer 5x5-Beispielmatrix durch!
- Skizzieren Sie das Zugriffsmuster des Algorithmus!

- Welche Laufzeit hat Cholesky und warum?
- Wie ist eine iterative Methode (bei uns) im Grundsatz definiert?
- Wie wird bei Jacobi die Inverse von A approximiert (und warum so)?
- Implementieren Sie den Jacobi-Algorithmus: Komponenten-basiert wie im Pseudocode und Matrix-Vektor-basiert
- Wie ist das Konvergenzverhalten von Jacobi (skizziert)? Wie schafft man (teilweise) Abhilfe?
- Optional, in die Zukunft gedacht: Wie lässt sich Jacobi parallelisieren?

15.6 Vorlesung VI

- Was sind die Grundideen hinter CG?
- Welche Operationen sind pro Iteration durchzuführen?
- Welcher Aufwand ergibt sich pro Iteration?
- Wie vergleicht sich CG mit Jacobi und Cholesky (bei typischen Eingaben)?
- Wie unterscheidet sich Nebenläufigkeit von Parallelität?
- Welche Kategorien von parallelen Systemen unterscheiden wir (hier) vorrangig?
- Auf welchen Weise kommunizieren Prozessoren / Recheneinheiten in diesen Systemen miteinander?
- Was sind die wesentlichen Performance-Kriterien für parallele Algorithmen? Erklären Sie diese!
- Warum ergibt Amdahls Gesetz eine obere Schranke für den Speedup?
- Sie wollen ein gegebenes sequentielles Programm (noch ohne Parallelisierung) beschleunigen. Warum sollten Sie sich gemäß Amdahls Gesetz auf die Teile konzentrieren, die besonders viel Laufzeit “fressen”?
- Was bedeutet Parallelisierbarkeit einer Schleife? Was muss dafür hinsichtlich der Datenabhängigkeiten gelten?
- Geben Sie ein Beispiel für eine Schleife mit/ohne Abhängigkeiten!

- Was ist eine Reduktionsoperation? Wie programmiert man sie prinzipiell? Ausblick: Wie macht man das parallel?

15.7 Vorlesung VII

- Welche Eigenschaften hat ein Parallelrechner mit verteiltem Speicher?
- Wie grenzt sich das von anderen Architekturmodellen ab?
- Was sind die Vorteile aus Anwendersicht, wenn man eine weit verbreitete Bibliothek zur parallelen Kommunikation nutzen kann?
- Was ist die Grundidee bei MPI-Programmen (konzeptionell – hinsichtlich der Aufteilung der Berechnung bzw. der Aufteilung der Daten)? Wie lässt sich das am Beispiel (SpMV oder andere Operationen) darstellen?
- Wir kennen erste Methoden zum Versenden und Empfangen von Nachrichten. Diese warten immer, bis sie "fertig" sind und setzen dann erst mit der Ausführung fort. Warum kann das je nach Einsatz ungünstig sein?
- Was ist der Vorteil von nicht-blockierender p2p-Kommunikation?
- Worauf ist dabei bzgl. Programmkorrektheit zu achten?
- Warum kann es auch Vorteile haben, blockierend zu senden?

15.8 Vorlesung VIII

- Was sind die Vorteile von kollektiver Kommunikation?
- Sehen Sie - zumindest zum bisherigen Kenntnisstand - auch Nachteile?
- Wenn Sie MPI_Scatter mit P2P-Routinen nachbauen müssten: wie setzen Sie die Parameter im Aufruf von MPI_Send (bezogen auf die von MPI_Scatter)?
- Reimplementieren Sie aus Vorlesung und Übung bekannte P2P-basierte Algorithmen, aber nun mit kollektiver Kommunikation (wo angebracht)!
- Überlegen Sie sich eine geeignete Datenstruktur für eine dichte Matrix, die verteilt über die MPI-Prozesse gespeichert ist.
- Implementieren Sie grundlegende Funktionen der linearen Algebra für parallel/-verteilt gespeicherte Matrizen und Vektoren.

15.9 Vorlesung IX

- Implementieren Sie die vorgenannten Algorithmen mit MPI!
- Betrachten Sie (schematisch) verschiedene Operationen der LA und wie man sie verteilt parallel umsetzen müsste (Blockschema). Unter welchen Bedingungen ist eine vertikale Matrix-Aufteilung vorteilhaft, wann eine horizontale?
- Warum ist es wichtig, beim parallelen CG die meisten Nichtnulleinträge der Matrix in der Nähe der Hauptdiagonalen (= im Diagonalblock) zu haben?
- Wie korrespondieren Matrix und Graph miteinander bei der Beurteilung des Kommunikationsvolumens?
- Erweiterung (falls diskutiert in der Vorlesung): Was ist der potentielle Unterschied zwischen dem Kantenschnitt und dem Kommunikationsvolumen?
- Erläutern Sie das Konzept der Halo-Knoten! Wofür benötigt man es?
- Was ist beim Halo-Austausch hinsichtlich der Reihenfolge der Austauschoperationen zu beachten? Wie modelliert man das Problem?
- Erklären Sie, wie man CSR (bzw. CSC) anpassen kann für verteilt gespeicherte dünn besetzte Matrizen!
- Wie überlappt man Kommunikation und Berechnung bei SpMV?

15.10 Vorlesung X

- Machen Sie sich noch mal das Vorgehen mit 1D-Layout klar, auch die Korrespondenz zwischen Matrix und Graph!
- Welches Ziel der Partitionierung wird bei komplexen Netzwerken häufig nicht erreicht? Warum nicht? Welche Folge hat das für die parallele Ausführung?
- Wie ist SpMV mit 2D-Partitionierung durchzuführen? Geben Sie zunächst die grobe Vorgehensweise, dann konkrete MPI-Routinen an!
- Warum/wozu betreibt man Präkonditionierung?
- Was ist das grundsätzliche Vorgehen?
- Was ist zu beachten, wenn der numerische Löser eine symmetrische Matrix voraussetzt? Wie setzt man das um?

- Was muss man im CG ersetzen, um das vorläufige Verfahren zu erhalten? Wie leitet sich daraus dann PCG her?

15.11 Vorlesung XI

- Erläutern Sie das Modell des Zufallssurfers!
- Wie wird das Modell mathematisch modelliert? Schreiben Sie Matrizen und Vektoren hin und erläutern Sie deren Bedeutung!
- Wie funktioniert die Potenzmethode?
- Wie hängen Potenzmethode und „Simulation“ des Zufallssurfers zusammen?
- Wie lässt sich Konvergenz der Potenzmethode zeigen? Wodurch ist die Konvergenzrate bestimmt und warum?
- Welche Voraussetzungen gibt es, um Eigenlöser XY einzusetzen?
  Bzgl.: Matrix, ges. EV/EW, Laufzeit, Speicherverbrauch

15.12 Vorlesung XII

- Was ist die Kernidee der Lanczos-Methode? Welche Eigenschaft haben die so erzeugten Vektoren?
- Geben Sie die (drei Terme der) Rekurrenz-Gleichung an!
- Wie überträgt sich dies nun in den Pseudocode der LanczosMethode?
- Was haben parallele SpMV-Routinen mit Graphpartitionierung zu tun?
- Was sind Kantenschnitt, Randknoten, Kommunikationsvolumen?
- Wo liegen die Unterschiede zwischen den drei Maßen?
- Warum ist die Maximumsnorm über alle Blöcke hier in der Regel sinnvoller als die Summennorm?
- Wie arithmetisiert *(in Matrix- und Vektorschreibweise ausdrücken)* man den Kantenschnitt?
- Was ist die wesentliche Operation bei der spektralen Partitionierung?

- Wie leitet man diesen Ansatz her? Was sind die einzelnen Schritte?
- Wie sind Laufzeit und Qualität einzuschätzen?
- Was sind die Vor- und Nachteile von spektraler Partitionierung?

15.13 Vorlesung XIII

- Was ist das wesentliche Kernelement bei einer lokalen Suche? Geben Sie ein konkretes Beispiel für dieses "Kernelements" bei GPP!
- Welche Vorteile/Nachteile haben globale Verfahren zur Optimierung, welche haben lokale? Erläutern Sie dies konkret für das GPP.
- Was ist die Stärke eines Mehrebenen-Verfahrens im Hinblick auf lokale vs globale Optimierung? Wann funktioniert es üblicherweise gut?
- Welche Komponenten benötigt ein Mehrebenen-Verfahren?
- Was sind übliche Kontraktionskerne, wie wird mit ihnen vergröbert?
- Was ist ein Matching? Warum berechnen wir ein gewichtetes Matching zur Bestimmung von Kontraktionskernen für Multilevel-GPP? Wo kommen die Gewichte her und warum wollen wir maximieren?
- Wie funktioniert der PathGrowing-Algorithmus? Welche Laufzeit hat er? Welche Approx.güte? Beweisen Sie beide Aussagen!

15.14 Vorlesung XIV

- Was ist die grundlegende Idee hinter Label Propagation für GP?
- Wie macht man aus dieser Idee eine vollständige lokale Suche?
- Wann bricht man ab? Welche Strategien sind dafür denkbar?
- Was muss bei LP geändert werden, wenn man damit vergröbern will?
- Überlegen Sie sich, welche Feinheiten für die Implementierung einer konkreten Multilevel-Methode noch fehlen!
- Welche der besprochenen Techniken eignen sich für welche Arten von Graphen (nicht)? Warum (nicht)?

- Begründen Sie die Laufzeit der inneren Schleife $\mathcal{O}(|E|)$ von LP!
- Inwiefern und warum ähneln sich LP und SpMV?
- Wiederholen Sie die parallele Implementierung von SpMV (1D- und 2D-Layout)!
- Wie können Sie LP parallel implementieren?
- Inwiefern greift die Modellierung mit Graphpartitionierung zur Optimierung der Kommunikation zu kurz, wenn die Kommunikationskosten zwischen Prozessoren nicht uniform sind?
- Wie muss man die Problemformulierung entsprechend anpassen?