Softwarepraktikum "Parallele Numerik"

Abschlussvortrag

Rebecca Seelos, Alexander Lüngen, Joshua

28. August 2019

Gegeben

- Maße der Hertplatte
- Anfangstemperatur
- Randtemperatur
- Wärmezufuhr

Gesucht

- Wärmeverteilung
- Wärmeentwicklung über Zeit
- Zeit bis zu einer bestimmten Temperatur

Teilaufgaben

Finden einer geeigneten ...

- ... Methodik zum Lösen von Gleichungen
- ... Parallelisierungsmethode
- ... Programmiersprache zur Implementierung

Gauß-Seidel Verfahren - Motivation

Im Rahmen der Finitien Elemente Methode müssen häufig Gleichungen gelöst werden:
$\rightarrow$ *WelcheGleichungen*?
$\rightarrow$ *Gauß* – *Seidel* – *Verfahren*
Iteratives Verfahren um lineare Gleichungen näherungsweise zu lösen

Gauß-Seidel Verfahren - Anwendung

1. Gegeben das Gleichungssystem:

$$\begin{aligned} a_{11} * x_1 + a_{12} * x_3 + \ldots + a_{1n} * x_n &= b_1 \\ \vdots \qquad\quad \vdots \qquad\quad \vdots & \\ a_{n1} * x_1 + \quad \ldots \quad + a_{nn} * x_n &= b_n \end{aligned}$$

2. Wähle Startvektor:

$$x^0 \in \mathbb{R}^n$$

3. Iteriere über Vektoreinträge einen Schritt mit der Vorschrift:

$$\forall k = 0, 1, \ldots; \quad \forall j = 1, .., n:$$

$$x_j^{k+1} = \frac{1}{a_{j,j}} \left(b_j - \sum_{i=1}^{j-1} a_{j,i} x_i^{k+1} - \sum_{i=j+1}^{n} a_{j,i} x_i^k \right)$$

Parallelisierungsmethoden

Parallelisierung des Gauss-Seidel-Verfahrens

1. Wavefront
2. Diamondtiling
3. Jacobi-Iteration

Parallelisierungsmethoden - Wavefront

Problem: - unregelmäßiger Parallelisierungsgrad
Optimierungsmöglichkeit: - Freigabe von zuvor berechneten elementen für den nächsten Schritt

Parallelisierungsmethoden - Diamondtiling

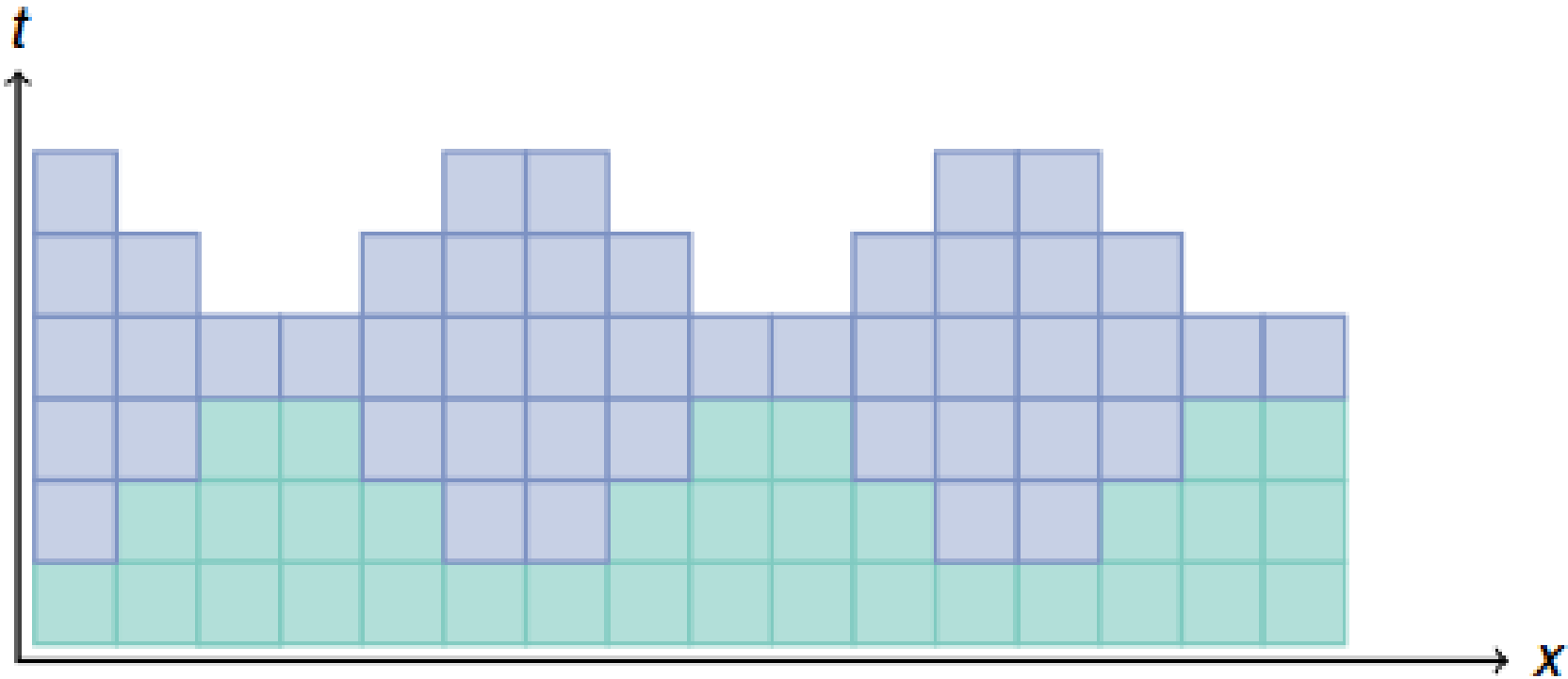


Abbildung: Quelle: Vorlesung "Software Engineering für moderne parallele Plattformen", Foliensatz "2.Entwurf"

Parallelisierungsmethoden - Jacobi-Iteration

Verändern der Rechenvorschrift von Gauß-Seidel-Verfahren
Neue Rechenvorschrift:
$x^{k+1} = \frac{1}{a_{i,i}()}$

Datenabhängigkeit nur im gleichen Zeitschritt

Wähle optimal bzgl. Speicher vs. Parallelisierungsgrad

OpenMP

Beispiele aus dem Praktikum: Numerische Berechnung von π, Mandelbrot, Gauss-Seidel
Die Vorteile sind:

- einfach in existierenden Code zu integrieren mit entsprechenden '#pragma'
- mit dem gcc compiler zu nutzen

OpenMP Nachteile

- Korrekte Anwendung wichtig
- verleitet eventuell dazu Dinge zu einfach zu sehen

Beispiel: Gauß-Seidel

Implementierung	l,h	T(1)	T(n)(parallel)	S(n)
OMP naiv	l=5, h=1/32	1.42	1.154	1.231
OMP naiv	l=6; h=1/64	37.216s	26.884s	1.384
OMP Jacobi	l=5, h=1/3	-	1.475s	0.963
OMP Jacobi	l=6, h=1/64	-	6.394s	5.82

CUDA

Vorteile:

- Nutzung der GPU (von NVIDIA)
- gut bei hoher Datenparallelität

Nachteile:

- bedarf Einarbeitung
- nvcc compiler, verschiedene Versionen von Grafikkarten, manchmal nicht kompatibel - angewiesen auf Hardware
- overhead

Beispiel: Vector inkrementieren

Zeit mit OpenMP auf CPU: 488.875 ms

	Speedup	
blocksize	Time	N = 10^8
4	141.23ms	3.49
32	18.020ms	27.74

Beispiel: Gauß-Seidel

Implementierung	l,h	T(1)	T(n)	S(n)
OMP naiv	l=5, h=1/32	1.42s	1.154s	1.231
OMP naiv	l=6; h=1/64	37.216s	26.884s	1.384
OMP Jacobi	l=5, h=1/3	-	1.475s	0.963
OMP Jacobi	l=6, h=1/64	-	6.394s	5.82
CUDA Jacobi	l=5, h=1/32	-	1.055s	1.346
CUDA Jacobi	l=6, h=1/64	-	1.318s	28.237

Finite-Differenzen-Methode

$$\frac{1}{h^2}\left(\begin{array}{cccc|cccc|cccc|cccc}
4 & -1 & & & -1 & & & & & & & & & & & \\
-1 & 4 & -1 & & & -1 & & & & & & & & & & \\
 & -1 & 4 & -1 & & & -1 & & & & & & & & & \\
 & & -1 & 4 & & & & -1 & & & & & & & & \\
\hline
-1 & & & & 4 & -1 & & & -1 & & & & & & & \\
 & -1 & & & -1 & 4 & -1 & & & -1 & & & & & & \\
 & & -1 & & & -1 & 4 & -1 & & & -1 & & & & & \\
 & & & -1 & & & -1 & 4 & & & & -1 & & & & \\
\hline
 & & & & -1 & & & & 4 & -1 & & & -1 & & & \\
 & & & & & -1 & & & -1 & 4 & -1 & & & -1 & & \\
 & & & & & & -1 & & & -1 & 4 & -1 & & & -1 & \\
 & & & & & & & -1 & & & -1 & 4 & & & & -1 \\
\hline
 & & & & & & & & -1 & & & & 4 & -1 & & \\
 & & & & & & & & & -1 & & & -1 & 4 & -1 & \\
 & & & & & & & & & & -1 & & & -1 & 4 & -1 \\
 & & & & & & & & & & & -1 & & & -1 & 4
\end{array}\right)\left(\begin{array}{c} u_1 \\ u_2 \\ u_3 \\ u_4 \\ \hline u_5 \\ u_6 \\ u_7 \\ u_8 \\ \hline u_9 \\ u_{10} \\ u_{11} \\ u_{12} \\ \hline u_{13} \\ u_{14} \\ u_{15} \\ u_{16} \end{array}\right) = \left(\begin{array}{c} f_1 \\ f_2 \\ f_3 \\ f_4 \\ \hline f_5 \\ f_6 \\ f_7 \\ f_8 \\ \hline f_9 \\ f_{10} \\ f_{11} \\ f_{12} \\ \hline f_{13} \\ f_{14} \\ f_{15} \\ f_{16} \end{array}\right)$$

Krylow-Unterraumverfahren & GMRES

- iterative Verfahren zum Lösen großer, dünnbesetzter Gleichungssysteme
- GMRES = Generalized minimal residual method
- Begrenzte Anzahl der Schritte bis zum Konvergenz
- Residuum abschätzen um Operationen zu sparen

Vergleich GMRES - Gauss-Seidel-Verfahren

-	Gauss-Seidel(s)	GMRES(s)	Speedup
l=5; h=1/32	0.488	0.025	19.52
l=6; h=1/64	0.960	0.135	7.11
l=7; h=1/128	8.856	1.300	6.812

LU-Zerlegung

- Weitere Verbesserungen durch Vorkonditionierung
- $A = (L * U)^{-1}$
- GMRES konvergiert schneller

Rand- und Anfangsbedingungen

- $-\triangle u(x, y, t_n) = \frac{f(x,y)-u'(x,y,t_n)}{a}$, $(x, y) \in \Omega = (0, 1)^2$, $n \in N \backslash 0$;
- $u(x, y, t_n) = 20$, $(x, y) \in \Gamma$, $n \in N$;
- $u(x, y, t_0) = 20$, $(x, y) \in \Omega$;
- f(x,y) ist stetig, $(x, y) \in \Omega$.

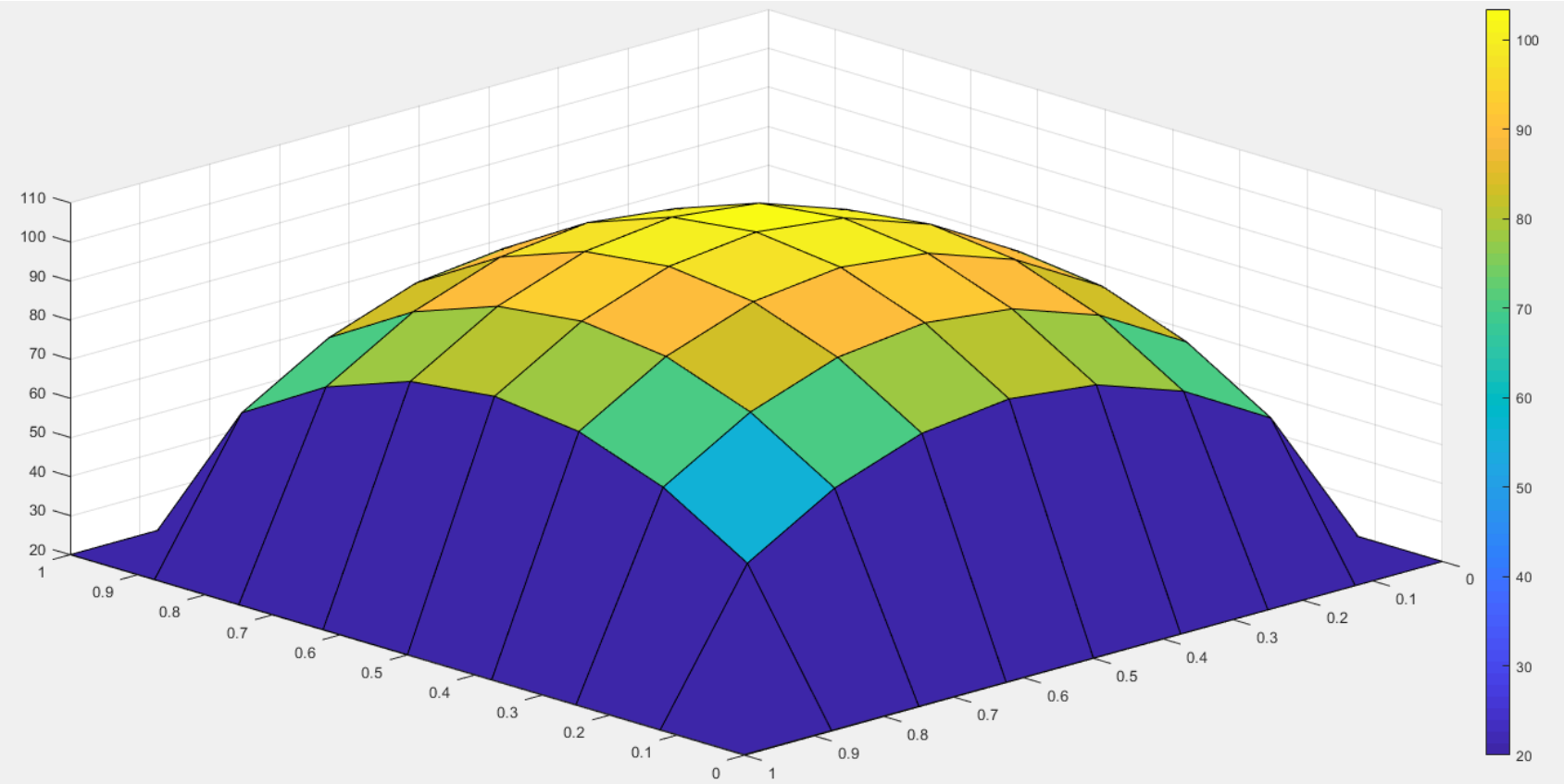
Zeitschritt

- $u'(t_n) = f + a * \triangle u(t_n)$
  mit $\triangle u(t_n) = \frac{4*u_{i,j}(t_n) - u_{i-1,j}(t_n) - u_{i+1,j}(t_n) - u_{i,j-1}(t_n) - u_{i,j+1}(t_n)}{h_s^2}$
- $u(t_{n+1}) = u(t_n) + h_t * u'(t_n)$
- $A * u(t_n) = \frac{h_s^2}{a} * (f - \frac{u(t_n) - u(t_{n-1})}{h_t})$

Implementierung

- OpenMp
- Vorkonditioniert
- GMRES (mit Residuumschätzung)

Ergebnisse


110
100
90
80
70
60
50
40
30
20
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
100
90
80
70
60
50
40
30
20

Ausblick

- Simulation weiter verschnellern (CUDA - GPU)
- Beliebige Hertplattenformen zulassen (zB. Kreis)
- Mit der Simulation optimieren (Abbruchbedingung anpassen?)

Fragen?

Danke für Ihre Aufmerksamkeit!