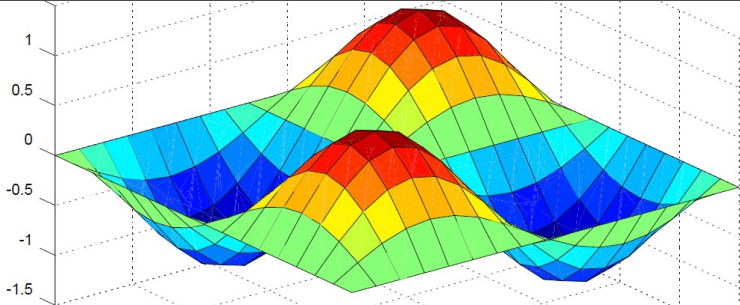


Projekt 1

Arbeiten mit OpenMP und FEM

Thore Mehr, Fabian Miltenberger, Sébastien Thill | 11.01.2017

LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELVERARBEITUNG (ITEC)



- 1 OpenMP und Tools (Aufgaben 1-2)
- 2 Parallelisierung (Aufgaben 3-4)
- 3 Partielle Differentialgleichungen (Aufgaben 5-6)

PI

- Reihenfolge der Threads nicht deterministisch
- Per Critical parallel langsamer als sequenziell
- Mit großen N und reduction Speedup nahe Kernzahl

Mandelbrot

- Speedup kleiner
- wegen Speicherverwaltung und Caches

- Meisten Fehler gefunden
- Durch Pragmas zu beheben

- O0-O3 sequenziell Pervormace ICC GCC
- O0-O3 parallel ICC schneller
- ICC fast viel schneller
- ICC macht größeren Binary x2
- ICC fast riesige Binary (30kb vs 1,4MB)
- äußere Schleife parallel am schnellsten

- Mit expliziten Chunksize etwa dynamic und static gleich schnell
- auto Chunksize mit static langsam
- je weniger Arbeit, desto stärker ist der Effekt
- zu große Chunksize sehr langsam

- Speedup $S(n)$

Der Geschwindigkeitszuwachs gegenüber einer sequentiellen Ausführung:

$$S(n) = \frac{T(1)}{T(n)}$$

- Efficiency $E(n)$

Beschleunigung pro Kern:

$$E(n) = \frac{S(n)}{n} = \frac{T(1)}{n \cdot T(n)}$$

Aufgabe 3 a)

- Mehraufwand $R(n)$

Durch Parallelisierung entstehender Aufwand:

$$R(n) = \frac{P(n)}{P(1)}$$

- Parallelindex $I(n)$

Operationen pro Zeiteinheit:

$$I(n) = \frac{P(n)}{T(n)}$$

- Auslastung $U(n)$

Operationen pro Zeiteinheit und Prozessor:

$$U(n) = \frac{I(n)}{n} = R(n) \cdot E(n) = \frac{P(n)}{n \cdot T(n)}$$

Navigationssymbole

Fehlerquellen durch Parallelisierung:

- Race Condition
Wettlaufsituationen, das Ergebnis hängt von konkreter Ausführungsreihenfolge ab
- Dead Lock
Verklemmung dadurch, dass Prozesse auf Freigabe von Ressourcen warten, die von anderen Prozessen gehalten werden, die ebenfalls warten
- Bibliotheken
- Messeffekte
- Cacheeffekte

Aufgabe 3 c)

Beschleuniger	Parallelität	Anwenderfreundlichkeit
CPU	Niedrig	Gut
GPU	Sehr hoch	Mittel
FPGA	Hoch	Abhängig von Bibliotheken
MIC	(Sehr) hoch	Gut

- **CPU:** Großer Instruktionssatz, Out-of-Order, selten mehr als 8 CPU Kerne, „gewöhnliche“ Programmierung
- **GPU:** Verbreitet, keine komplexen Befehle, massive Parallelität, *SIMD*, *CUDA* und *OpenCL*.
- **FPGA:** Beliebige Eigenschaften, Testen von Chips, „multifunktionelle“ Beschleunigungskarte, *OpenCL* oder spezielle Bibliotheken
- **MIC:** Gewöhnliche x86 Architektur, *OpenMP* oder *OpenCL*

◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶ ◀ ▶

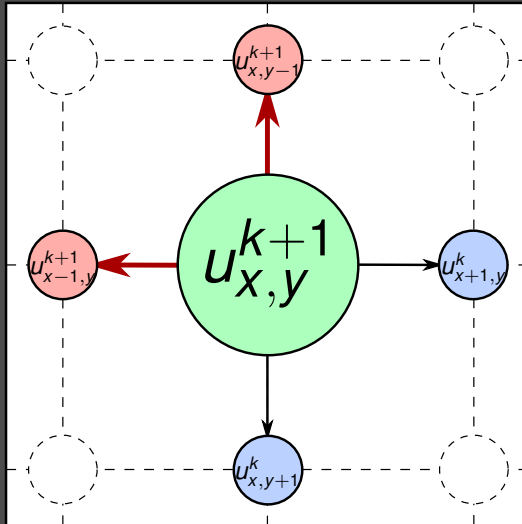
- Löst Gleichungssysteme der Form

$$Au = h^2 f$$

nach $u \in \mathbb{R}$ mit $A \in \mathbb{R}^{n \times n}$, $h \in \mathbb{R}$, $f \in \mathbb{R}^n$

- Bei uns: A , h Abhängigkeit von Gitterkonstante h gegeben, f abhängig von Problem
- A beschreibt Abhängigkeit eines Gitterpunkts zu 4 Nachbarnpunkten
 \Rightarrow dünn besetzt
- Problemgröße ist $n = m \times m$, mit m Seitenlänge des Gitters
- Iterativ, $u_{x,y}^k = u_j^k$ ist k -te Iterierte für Gitterpunkt x, y bzw. Eintrag j

Aufgabe 4 – Abhängigkeiten



Aufgabe 4 – Abhängigkeiten

Nach GSV:

$$u_j^{k+1} := \frac{1}{a_{j,j}}(h^2 f_j - \sum_{i=1}^{j-1} a_{j,i} u_i^{k+1} - \sum_{i=j+1}^n a_{j,i} u_i^k)$$

Zerfällt durch Struktur von A und Rand (mit $d := m - 2$) zu:

Pseudocode

```
double sum = h * h * f[j];  
if (j % d > 0) sum += u[j - 1]; // Linker Nachbarknoten  
if (j % d < d - 1) sum += u[j + 1]; // Rechter Nachbarknoten  
if (j / d > 0) sum += u[j - d]; // Oberer Nachbarknoten  
if (j / d < d - 1) sum += u[j + d]; // Unterer Nachbarknoten  
u[j] = sum / 4;
```

Annahme: Veränderung der Iterierten korreliert mit Abstand von Lösung
→ Breche ab, sobald Änderung von u^k zu u^{k+1} gering, also:

$$\|u^{k+1} - u^k\|_{\max} < \varepsilon_{\text{Error}}$$

Wir verwenden $\varepsilon_{\text{Error}} = 1 \times 10^{-6}$

Aufgabe 4 – Parallelisierung

Nacheinander, zuerst alle grauen Felder, dann die weißen:

$u_{1,1}^{k+1}$	$u_{2,1}^{k+1}$	$u_{3,1}^k$	$u_{4,1}^k$	$u_{5,1}^{k-1}$	$u_{6,1}^{k-1}$	$u_{7,1}^{k-2}$
$u_{1,2}^{k+1}$	$u_{2,2}^k$	$u_{3,2}^k$	$u_{4,2}^{k-1}$	$u_{5,2}^{k-1}$	$u_{6,2}^{k-2}$	$u_{7,2}^{k-2}$
$u_{1,3}^k$	$u_{2,3}^k$	$u_{3,3}^{k-1}$	$u_{4,3}^{k-1}$	$u_{5,3}^{k-2}$	$u_{6,3}^{k-2}$	$u_{7,3}^{k-3}$
$u_{1,4}^k$	$u_{2,4}^{k-1}$	$u_{3,4}^{k-1}$	$u_{4,4}^{k-2}$	$u_{5,4}^{k-2}$	$u_{6,4}^{k-3}$	$u_{7,4}^{k-3}$
$u_{1,5}^{k-1}$	$u_{2,5}^{k-1}$	$u_{3,5}^{k-2}$	$u_{4,5}^{k-2}$	$u_{5,5}^{k-3}$	$u_{6,5}^{k-3}$	$u_{7,5}^{k-4}$
$u_{1,6}^{k-1}$	$u_{2,6}^{k-2}$	$u_{3,6}^{k-2}$	$u_{4,6}^{k-3}$	$u_{5,6}^{k-3}$	$u_{6,6}^{k-4}$	$u_{7,6}^{k-4}$
$u_{1,7}^{k-2}$	$u_{2,7}^{k-2}$	$u_{3,7}^{k-3}$	$u_{4,7}^{k-3}$	$u_{5,7}^{k-4}$	$u_{6,7}^{k-4}$	$u_{7,7}^{k-5}$

Aufgabe 4 – Performance

Ermittelt auf dem *i82sn07* Rechner mit 32 Kernen:

Problem n	Seq. Zeit	2 Threads		4 Threads		...	32 Threads	
		T(2)	S(2)	T(4)	S(4)		T(32)	S(32)
225	0,005	0,007	0,357	0,008	0,625	...	1,361	0,0389
961	0,053	0,056	0,473	0,049	1,08	...	4,041	0,178
⋮	⋮	⋮		⋮		...	⋮	
65.025	115	81	1,42	45,3	2,54	...	25,33	4,54
261.121	1340	943	1,42	504	2,67	...	177	7,57

Gegeben:

$$-\Delta u(x, y) = f(x, y), (x, y) \in \Omega = (0, 1)^2, u(x, y) = 0, (x, y) \in \Gamma$$

Die Bedingungen:

- Ω beschränktes Gebiet
- Γ hinreichend glatt
- $f : \Omega \rightarrow \mathbb{R}$

Aufgabe 5 b)

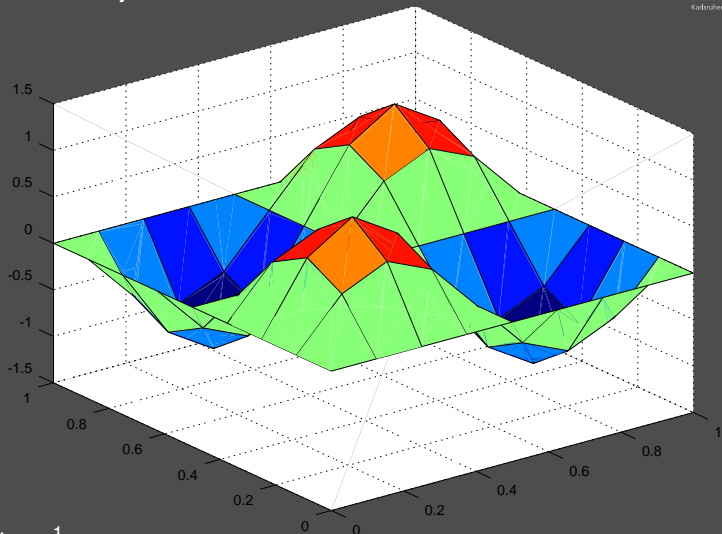
Gesucht ist f mit

$$u(x, y) = \sin(2M\pi x) \sin(2N\pi y)$$

Anwendung des *Laplace-Operators* Δ :

$$\begin{aligned} f(x, y) &= -\Delta u(x, y) \\ &= -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} \\ &= -\frac{\partial}{\partial x} (2M\pi \cos(2M\pi x) \sin(2N\pi y)) - \frac{\partial}{\partial y} (2N\pi \sin(2M\pi x) \cos(2N\pi y)) \\ &= 4M^2\pi^2 \sin(2M\pi x) \sin(2N\pi y) + 4N^2\pi^2 \sin(2M\pi x) \sin(2N\pi y) \\ &= (M^2 + N^2)4\pi^2 \sin(2M\pi x) \sin(2N\pi y) \end{aligned}$$

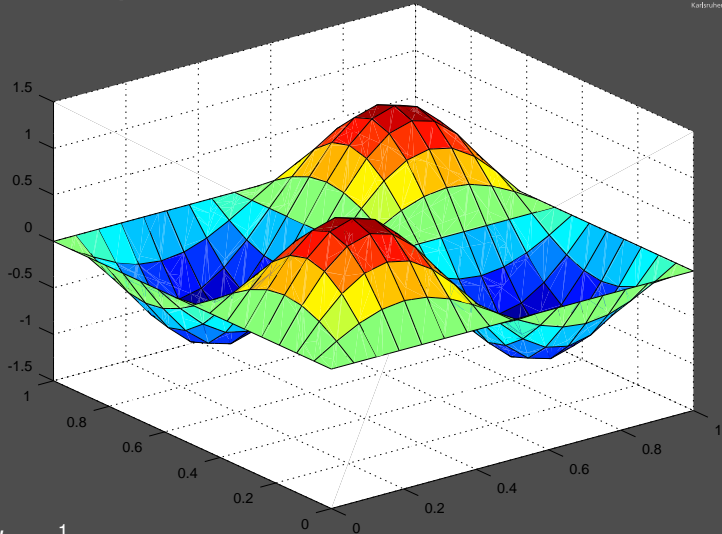
Aufgabe 5 c)



$$l = 3, h = \frac{1}{8}$$

Navigation icons: back, forward, search, etc.

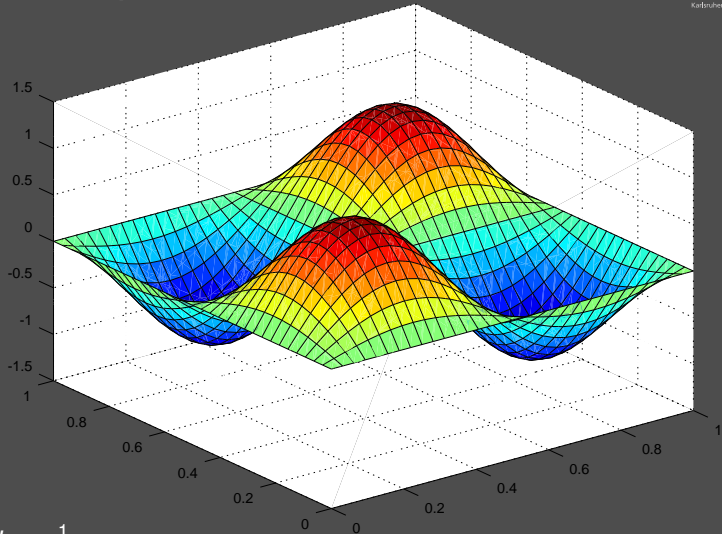
Aufgabe 5 c)



$$l = 4, h = \frac{1}{16}$$

Navigation icons: back, forward, search, etc.

Aufgabe 5 c)



$$l = 5, h = \frac{1}{32}$$

Navigation icons: back, forward, search, etc.

Fehler zu analytischen Lösung wird mit kleinerem h kleiner
 \Rightarrow Es handelt sich um eine *h-FEM-Methodik*, da die Polynomgrade nicht verändert wurden

Alphabetische Liste gängiger Krylow-Unterraum-Verfahren:

- Arnoldi-Verfahren, zur Eigenwertapproximation
- BiCG, das CG-Verfahren für nicht SPD-Matrizen
- BiCGSTAB, Stabilisierung von CGS
- BiCGSTAB(ell), Stabilisierung von CGS
- BiCGSTABTFQMR, der Ansatz hinter TFQMR angewandt auf BiCGSTAB
- BiOres, eine Variante des BiCG-Verfahrens
- BiOmin, eine Variante des BiCG-Verfahrens
- BiOdir, eine Variante des BiCG-Verfahrens
 - CG, zur approximativen Lösung linearer Gleichungssysteme
 - CGNE, CG-Verfahren auf den Normalgleichungen, Variante 1
 - CGNR, CG-Verfahren auf den Normalgleichungen, Variante 2
 - CGS-Verfahren, quadrierte BCG-Rekursion
 - CGS-Verfahren, quadrierte BCG-Rekursion
 - CGS-Verfahren, quadrierte BCG-Rekursion
 - CGS-Verfahren, quadrierte BCG-Rekursion
- ...

⇒ Es gibt sehr viele

Für uns relevant:

- CG-Verfahren
Geeignet für große lineare, symmetrische, positiv definite und dünn besetzte LGS, spätestens n Schritten exakte Lösung
- GMRES
Geeignet für große, dünn besetzte LGS, exakte Lösung erst nach endlich vielen Schritten
- Lanczos-Verfahren
Konvergenz von Eigenwerten abhängig

Aufgabe 6 b)

Speedup und Effizienz von *CG-Verfahren* gegenüber *Gauß-Seidel-Verfahren* (Problem aus Aufgabe 5)

Problem n	Seq.	2 Threads		...	16 Threads		32 Threads	
	S(1)	S(2)	E(2)	...	S(16)	E(16)	S(32)	E(32)
225	1,74	1,81	0,906	...	2,44	0,153	2,93	0,0917
961	4,01	3,71	1,86	...	1,44	0,0898	3,46	0,109
⋮	⋮	⋮		...	⋮		⋮	
65.025	17,9	16,5	8,27	...	16,8	1,05	4,05	0,127
261.121	27,6	25,9	12,9	...	39,5	2,47	37,9	1,18

Vorkonditionierung zerstört dünne Struktur von A

→ Nur sinnvoll, wenn A nicht dünn besetzt wäre

Aufgabe 6 c)

*HiFlow*³ würde sich eignen:

Ausgereifte OpenMP Funktionalität → verringerter Portierungsaufwand

Bibliothek	Unterschiede	Gemeinsamkeiten
HiFlow ³	Hohe Parallelität Keine externen Bibliotheken nötig	C++, Krylov- Unterraumverfahren, Diskretisierung wählbar
MFEM	Sehr hohe Parallelität (mehrere hunderttausend Kerne) Keine externen Bibliotheken nötig	
Deal.II	Hohe Parallelität (16000 Kerne mindestens), Keine externen Bibliotheken benötigt (optional Libraries)	

Mit *OpenMP* lässt es sich parallelisieren.