Antworten zu Aufgabenblatt 1

Alexander Lüngen

2019-06-04

1 Aufgabe 1

1.1 a)

Die Reihenfolge in welcher die IDs der Threads ausgegeben werden ist unterschiedlich und auch nicht vorherzusagen.

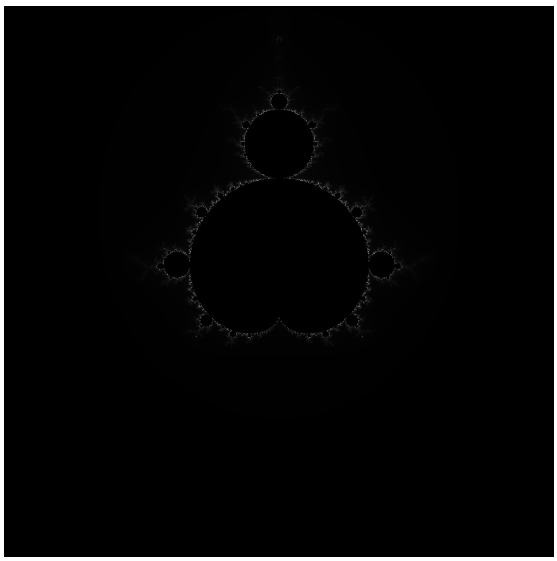
1.2 b)

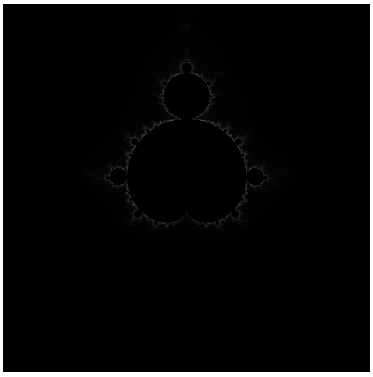
Gemessen wurde auf dem i82sn02. Die manuelle parallelisierung wurde mit atomic gemacht. Die Messung wurde nach 4 Threads aufgehört da sich die Rechenzeit sehr erhöht hat. Maximales Speedup von ~ 8.

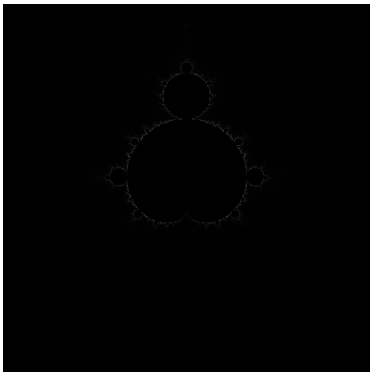
-	Number of Threads	N	Time (s)	Speedup
sequential	1	1000	0.002	-
	1	10^7	0.239	-
	1	10^8	2.388	-
	1	10^9	23.642	-
	1	10^{10}	239.508	-
manual	2	1000	0.002	1
	2	10^7	0.634	-
	2	10^8	6.461	-
	2	10^9	52.989	-
	4	10	0.003	-
	4	10^7	1.214	-
	4	10^8	13.491	-
	4	10^9	~ 120	-
reduction	2	1000	0.002	1
	2	10^7	0.121	1.9
	2	10^8	1.188	2
	2	10^9	11.845	1.9
	2	10^{10}	118.204	2
	4	1000	0.003	-
	4	10^7	0.062	3.9
	4	10^8	0.595	4
	4	10^9	5.959	3.9
	4	10^{10}	59.183	4
	8	1000	0.003	-
	8	10^7	0.032	7.5
	8	10^8	0.299	7.9
	8	10^9	2.967	7.9
	8	10^{10}	29.689	8.0
	16	1000	0.003	-
	16	10^7	0.040	5.9
	16	10^8	0.318	7.5
	16	10^9	3.028	7,8
	16	10^{10}	29.674	8.1
	32	1000	0.004	-
	32	10^7	0.037	6.4
	32	10^8	0.305	7,8
	32	10^9	2.987	7.9
	32	10^{10}	29.551	8.1

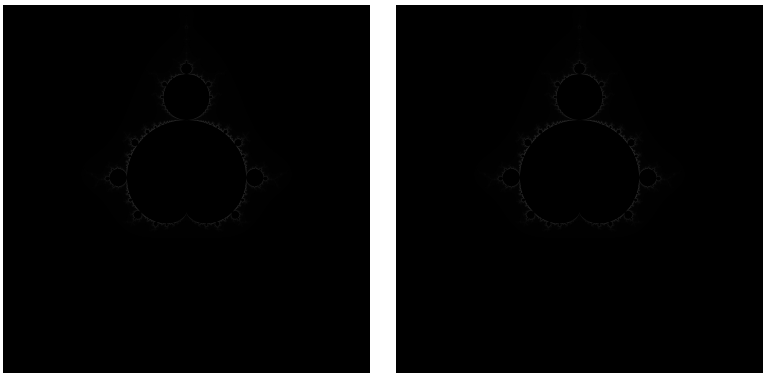
1.3 c)

-	Auflösung	Time (s)	Speedup
sequential	1000	0.002	-
	1	0.239	-
	1	2.388	-
	1	23.642	-
	1	239.508	-









2 Aufgabe 2

2.1 a)

1. Example: Es besteht eine Race-Condition auf dem Array a über Index i, bei ungünstiger Ausführungsreihenfolge der Threads kann es dazu kommen, das a[i] durch einen Thread geschrieben wird und von einem anderen Thread versucht wird, darauf in der nachfolgenden Anweisung lesend zuzugreifen. Um dieses Problem zu lösen können zwei for-Schleifen verwendet werden. Eine schreibt alle Werte in Array a und die zweite schreibt alle Werte unter Zugriff auf Array a in Array b. Beide Schleifen werden jeweils getrennt parallelisiert.

2. Example: Threads exisiteren hier in der gesamten parallelen Region. Das nowait-Statement der ersten parallelisierten For-Schleife bewirkt, dass Threads bereits die nächste parallelisierte Schleife bearbeiten können, wenn sie ihren Teil der ersten Schleife fertig verarbeitet haben. Dies führt zu einer Race-Condition auf das Array a mit Index i, ähnlich zu Example 1. Mit dem entfernen des nowait-Statements der ersten for-Schleife warten alle Threads wieder auf die implizite Barriere bis alle Threads fertig sind und es kann Problemlos auf die geschriebenen Werte in Array a zugegriffen werden.

3. Example: Hier ist die Variable x zunächst global definiert und wird somit implizit zwischen den Threads geshared. Somit besteht eine Race-Condition auf x da die Threads unabhängig voneinander sowohl lesend als auch schreibend auf x zugreifen. Indem man x explizit als private deklariert, erhält jeder Thread seine eigene Kopie der Variable und es besteht keine Race-Condition mehr.

4. Example: f ist global definiert und wird durch jeden Thread private gesetzt. Allerdings erhält jeder Thread eine uninitialiserte Kopie der Variable f, was bedeutet das dass initialisieren mit dem Wert 2 vor der Schleife nur für den Master-Thread sichtbar ist. Um diese Initialiserung auch für

alle übrigen Threads sichtbar zu machen, ist es erforderlich, f mittels firstprivate zu deklarieren. Des weiteren wird der Wert von x nicht aus der parallelen Region wieder rausgeschrieben, sondern gelöscht. Wodurch der zuletzt geschriebene Wert innerhalb der parallelen Region in x nach außen hin nicht sichtbar ist. Um deses Verhalten zu erreichen, muss x als lastprivate deklariert werden.

5. Example:

2.2 b)

3 Aufgabe 3

3.1 a)

P(x): Anzahl auszuführender Operationen auf x Prozessoren.
T(x): Ausführungszeit auf x Prozessoren.

Speedup: Der Zusammenhang zwischen serieller und paralleler Ausführungszeit eines Programmes. S(n) = T(1)/T(n)

Effizienz: E(n) = S(n)/n. Gibt die relative Verbesserung der Verarbeitungsgeschwindigkeit an.

Auslastung: $R(n)/(n*T(n))$. Gibt an, wie viele Operationen (Tasks) jeder Prozessor im Durchschnitt pro Zeiteinheit ausgeführt hat.

Mehraufwand: $R(n) = P(n)/P(1)$. Beschreibt den bei einem Multiprozessorsystem erforderlichen Mehraufwand für die Organisation, Synchronisation und Kommunikation der Prozessoren.

3.2 b)

Race-Conditions: Wenn zwei Threads unabhängig voneinander auf eine Ressource lesend oder auch schreibend zugreiffen können, spricht man von einer Race-Condition. Hierbei kann es bspw. beim Zugriff auf Variablen bei ungünstiger Ausführungszeiten dazu kommen, dass am Ende der Berechnungen ein falscher Wert in der Variable enthalten ist, als wenn die Berechnung sequentiell ausgeführt worden wäre. Um eine Race-Condition zu vermeiden, können die kritischen Abschnitte in der Art und weise gesichert werden, das immer nur ein Thread gleichzeitig innerhalb des kritischen Abschnitts sein darf.

In diesem Zusammenhang kann es auch zu Deadlocks, Lifelocks oder auch Starvation kommen.

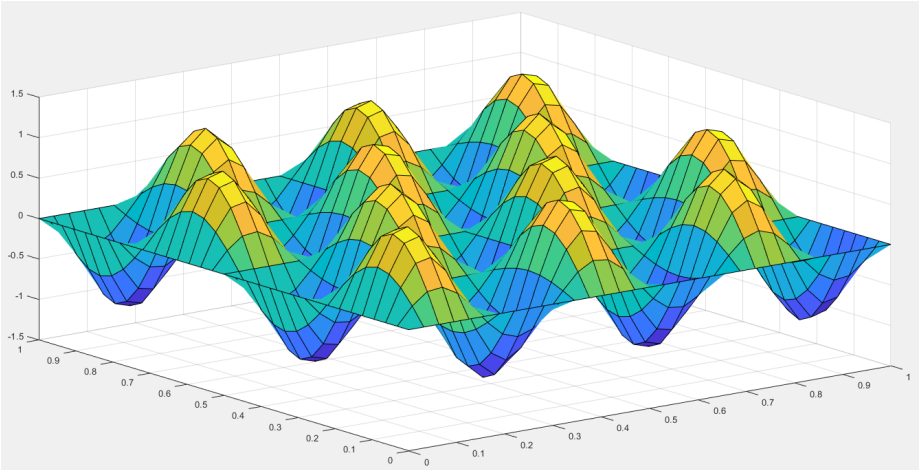


Figure 1: l=5; h=1/32;

3.3 c)

-	GPUs	CPUs	FPGAs
Energieeffizienz	Gut	Mittel	Gut
Anwenderfreundlichkeit	• Braucht Einarbeitungszeit. • Es gibt Bibliotheken.	• Am einfachsten zu programmieren. • Viele Bibliotheken vorhanden. • Kurze Compilierzeit.	• Aufwändig zu programmieren. • Lange Compilierzeit. • Wenig Bibliotheken.

In diesem Praktikum verwenden wir CPUs und GPUs da sie universeller einsetzbar sind als FPGAs und die Programmierung deutlich leichter ist.

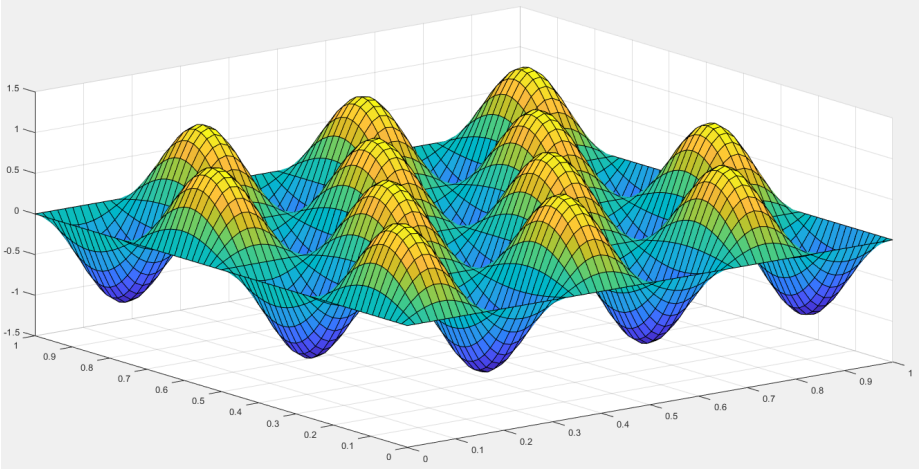


Figure 2: l=6; h=1/64;

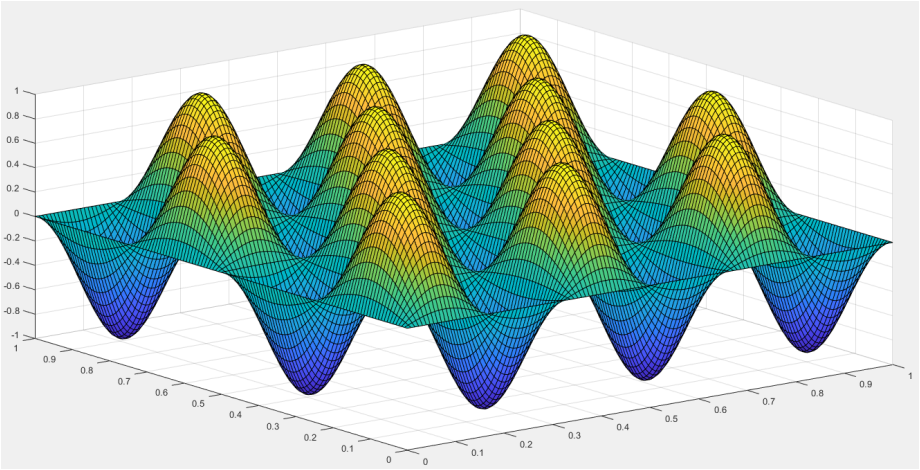


Figure 3: l=7; h=1/128;

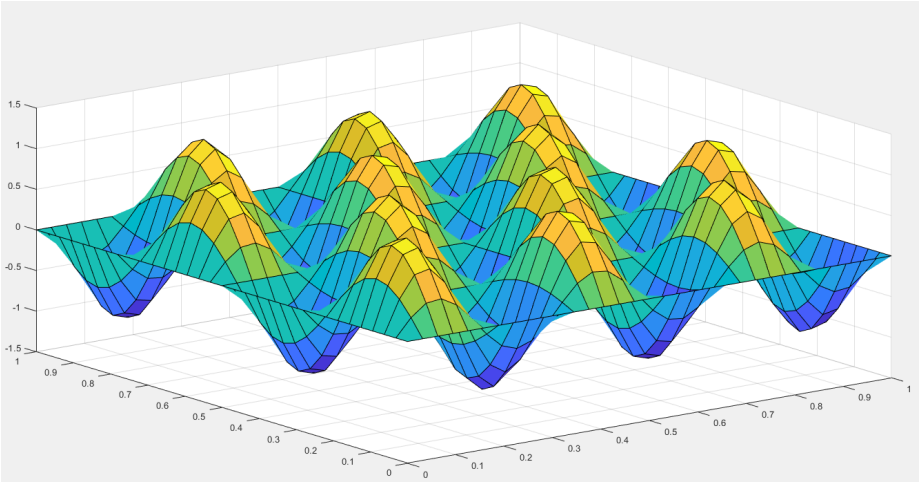


Figure 4: GMRES; l=5; h=1/32;

4 Aufgabe 4

5 Aufgabe 5

5.1 a)

f(x,y) muss definiert sein auf Ω
Γ ist der Rand von Ω
Lösung u(x,y) muss zweifach differenzierbar sein

5.2 b)

f(x,y) = $(N^2 + M^2)$*4*π^2*sin(2*M*π*x)*sin(2*N*π*y)

5.3 c)

Es handelt sich um eine h-FEM.
Die Lösungswerte in Figure 1, 2 und 3 ergrben sich mit M=3; N=2; l=5,6,7;

6 Aufgabe 6

6.1 a)

Liste: CG Verfahren; PCG-Verfahren; Verfahren der minimalen Risiduen(GMRES); GCR-Verfahren; Arnoldi-Verfahren; FOM, ORTHORES;
Das Gleichungssystem ist dünnbesetzt und alle Krylow-Unterraumverfahren sind

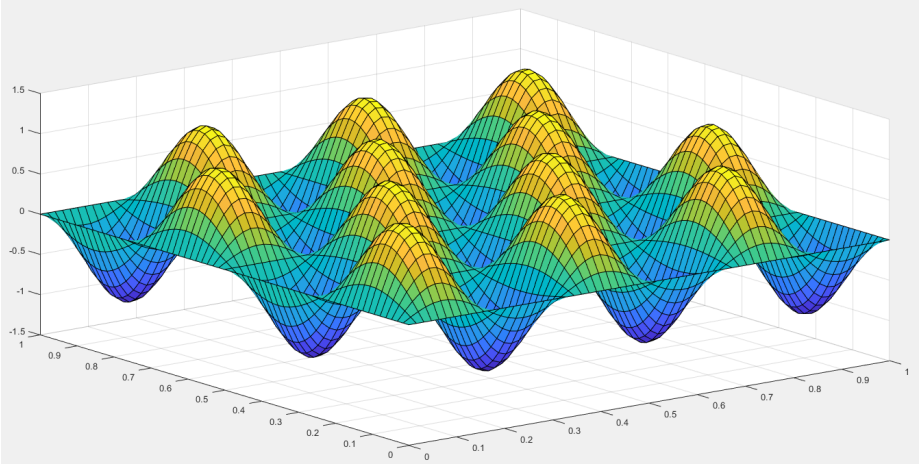


Figure 5: GMRES; l=6; h=1/64;

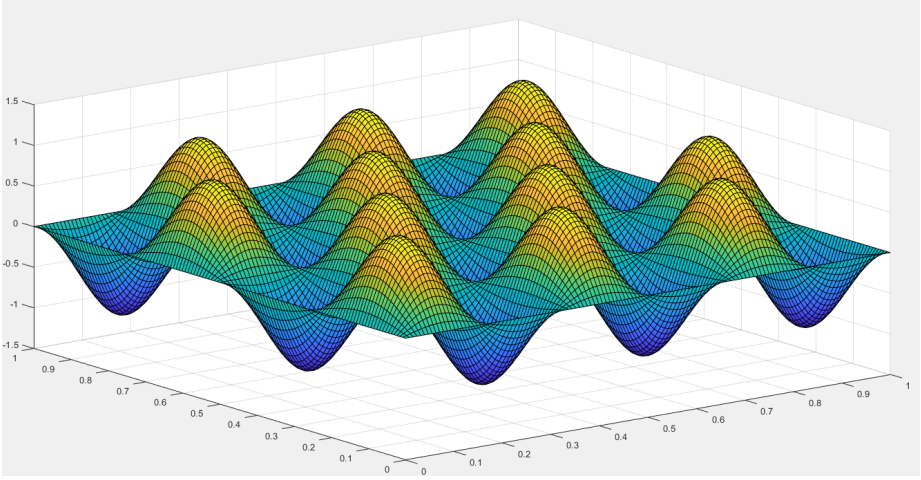


Figure 6: GMRES; l=7; h=1/128;

gut geeignet für dünnvesetzte Gleichungssysteme. Diese Lösungswerte in Figure 4, 5 und 6 ergeben sich mit der Gleichung aus Aufgabe 5 und M=3; N=2; l=5,6,7;

6.2 b)

Die Verwendung des Residuums als Abrruchbedingung ist eventuell problematisch, da man dadurch in jedem Schleifendurchlauf einen Lösungsvektor x^k berechnen muss. Man kann sich eine feste Anzahl von Iterationen setzen, allerdings hat man dann keine garantierte Genauigkeit. Außerdem kann man das Residuum mit weniger Rechenaufwand in jeder Iteration abschätzen und diesen Schätzwert als Abbruchbedingung nutzen. Auch diese Vorgehensweise hat keine garantierte Genauigkeit, da die Schätzgenauigkeit variiert. Im Zuge des Praktikums haben wir bis jetzt nur eine stabile Version mit abgeschätztem Residuum implementiert. Die Version mit dem Residuum als Abbruchbedingung funktioniert noch nicht verlässlich.

6.3 c)

Die Efficiency ist bei jedem Vergleich zwischen Aufgabe 6 und Aufgabe 5c gleich dem Speedup, da die Anzahl N der Kerne auf denen das Programm läuft nicht variiert.

-	Aufgabe 5(s)	Aufgabe 6(s)	Speedup
l=5; h=1/32	+1.450	+0.028	56.96
	+1.420	+0.021	
	+1.371	+0.026	
	+1.475	+0.026	
	+1.408	+0.026	
	=1.424	=0.025	
l=6; h=1/64	+15.427	+0.140	98.488
	+12.546	+0.148	
	+12.561	+0.089	
	+12.766	+0.142	
	+13.081	+0.155	
	=13.276	=0.135	
l=7; h=1/128	+107.183	+1.289	81.113
	+105.581	+1.188	
	+104.562	+1.175	
	+105.834	+1.703	
	+104.077	+1.148	
	=105.447	=1.300	

Einsatz eines Vorkonditionierers: Der Einsatz ist unsinnvoll, da wir bereits eine dünn besetzte Matrix gegeben haben. Falls die Matrix anders besetzt wäre, könnte ein Vorkonditionierer von Nutzen sein.

6.4 d)

MFEM; deal.II; libMesh; JuliaFEM; FEniCS; Hermes Project;...
Liste weiterer FEM Packages/Libraries: https://en.wikipedia.org/wiki/List_of_finite_element_software_packages
Fast alle sind Open Source und kostenlos nutzbar. "Hermes Project" scheint
zum Beispiel eine leicht nutzbare C/C++ Bibliothek.

7 Aufgabe 7