Antworten zu Aufgabenblatt 1

Alexander Lüngen

2019-06-04

1 Aufgabe 1

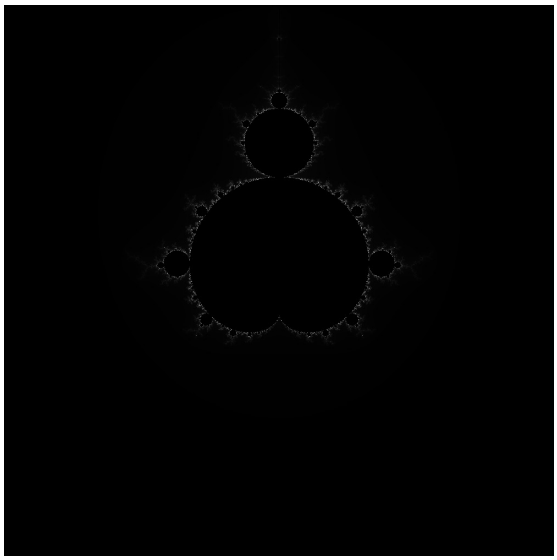
1.1 a)

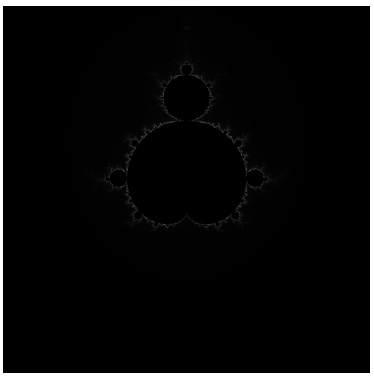
Die Reihenfolge in welcher die IDs der Threads ausgegeben werden ist nicht immer gleich.

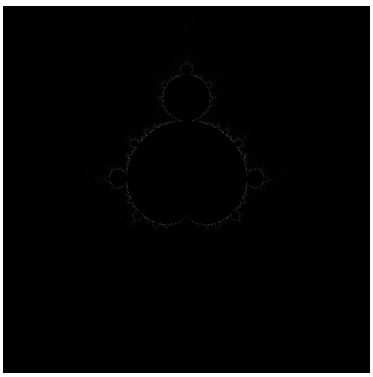
1.2 b)

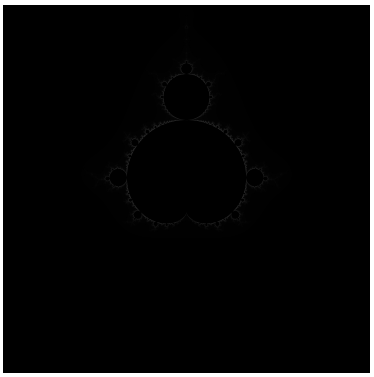
-	Number of Threads	N	Time	Speedup
manual	2	10	0:00	0.8
	2	100	0:00	0.8
	2	1000	0:00	0.8
	2	10000	0:00	0.8
	4	10	0:00	0.8
	4	100	0:00	0.8
	4	1000	0:00	0.8
	4	10000	0:00	0.8
	8	10	0:00	0.8
	8	100	0:00	0.8
	8	1000	0:00	0.8
	8	10000	0:00	0.8
	16	10	0:00	0.8
	16	100	0:00	0.8
	16	1000	0:00	0.8
	16	10000	0:00	0.8
	32	10	0:00	0.8
	32	100	0:00	0.8
	32	1000	0:00	0.8
	32	10000	0:00	0.8
reductino	2	10	0:00	0.8
	2	100	0:00	0.8
	2	1000	0:00	0.8
	2	10000	0:00	0.8
	4	10	0:00	0.8
	4	100	0:00	0.8
	4	1000	0:00	0.8
	4	10000	0:00	0.8
	8	10	0:00	0.8
	8	100	0:00	0.8
	8	1000	0:00	0.8
	8	10000	0:00	0.8
	16	10	0:00	0.8
	16	100	0:00	0.8
	16	1000	0:00	0.8
	16	10000	0:00	0.8
	32	10	0:00	0.8
	32	100	0:00	0.8
	32	1000	0:00	0.8
	32	10000	0:00	0.8

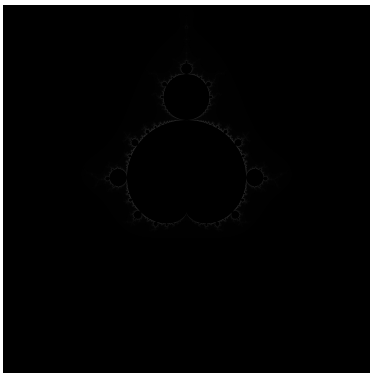
1.3 c)











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)

Speedup: Effizienz:

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.

3.3 c)

CPU GPGPU FPGA MICs

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;

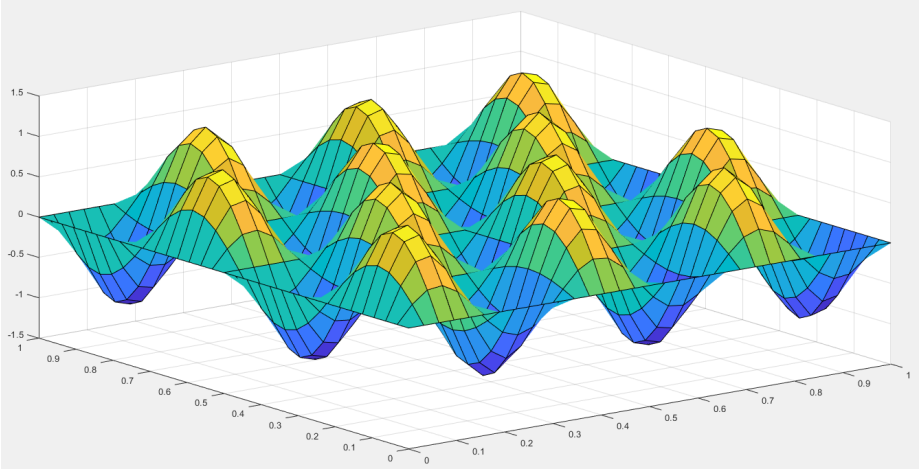


Figure 1: l=5; h=1/32;

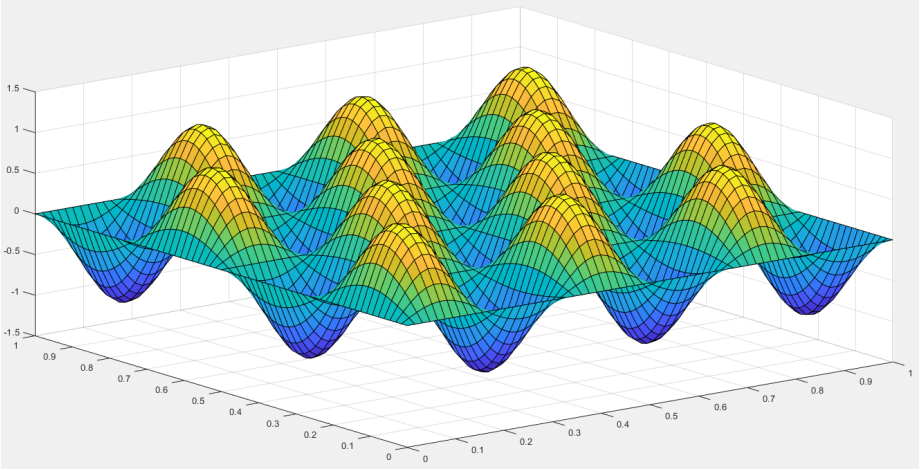


Figure 2: l=6; h=1/64;

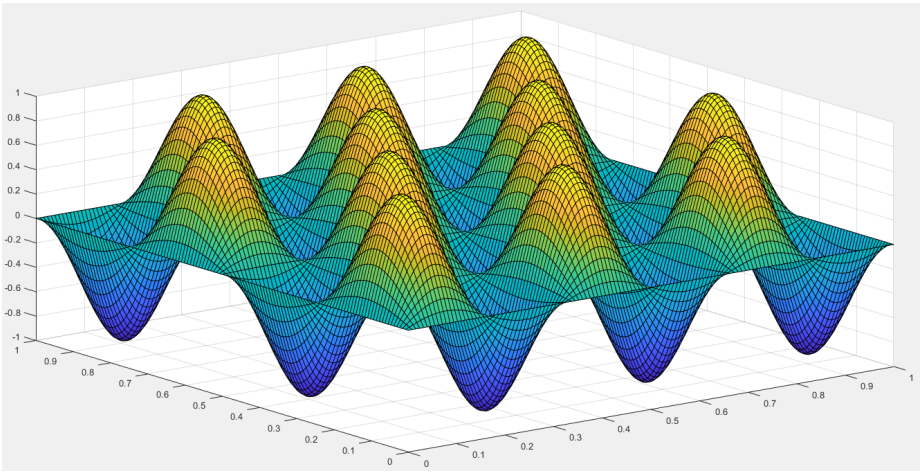


Figure 3: l=7; h=1/128;

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 zbd alle Krylow-Unterraumverfahren sind 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.

6.3 c)

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.

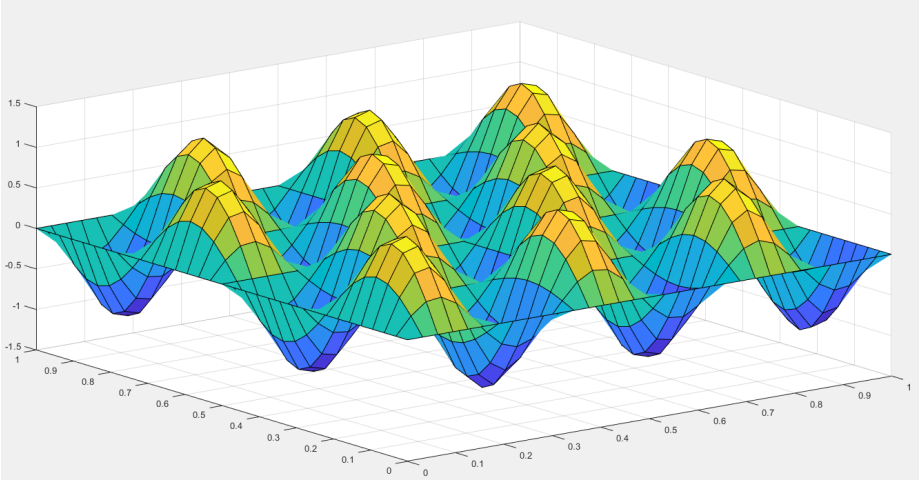


Figure 4: GMRES; l=5; h=1/32;

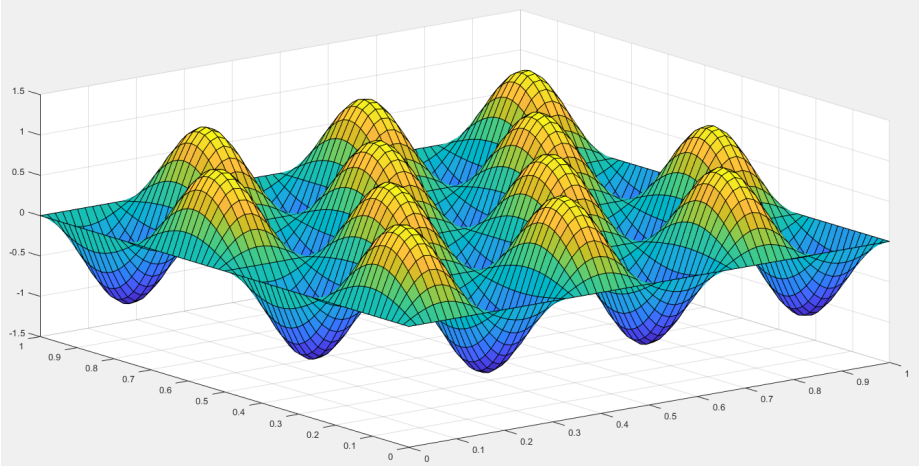


Figure 5: GMRES; l=6; h=1/64;

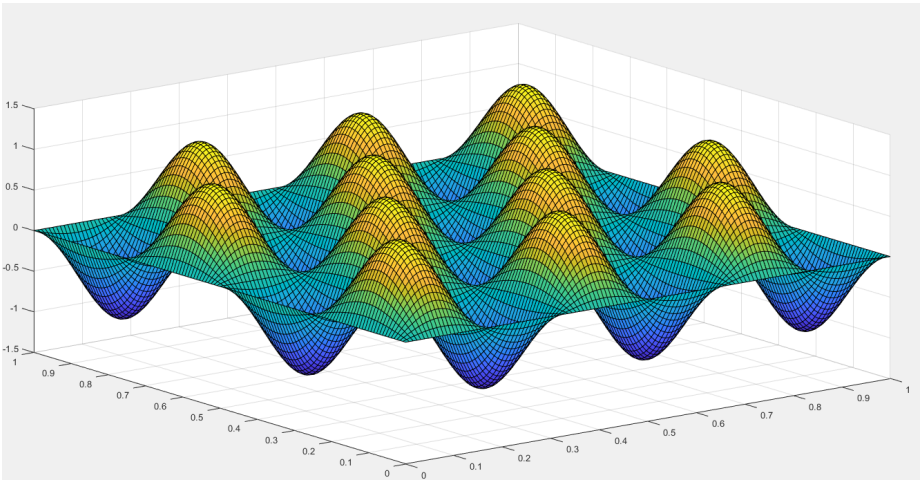


Figure 6: GMRES; l=7; h=1/128;

6.4 d)

7 Aufgabe 7