

Antworten zu Aufgabenblatt 1

Alexander Längen

2019-06-04

1 Aufgabe 1

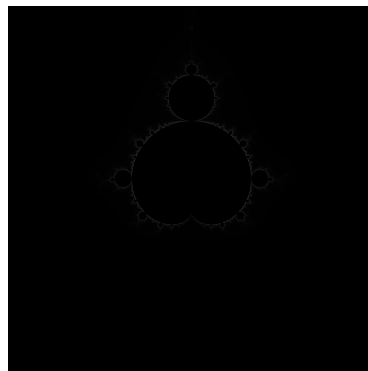
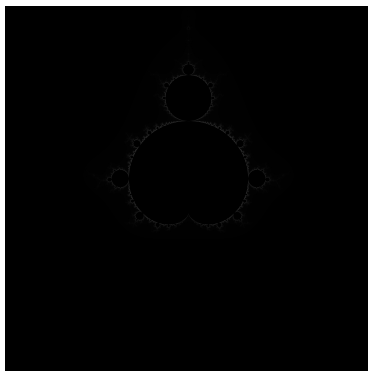
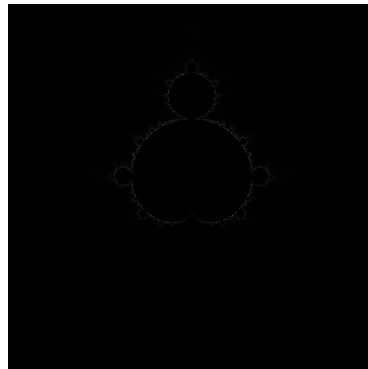
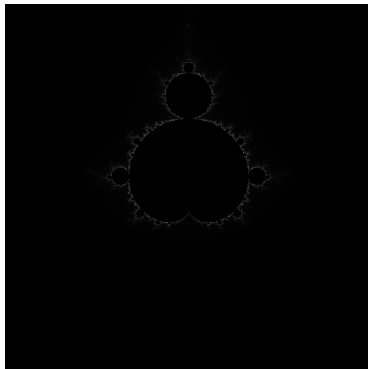
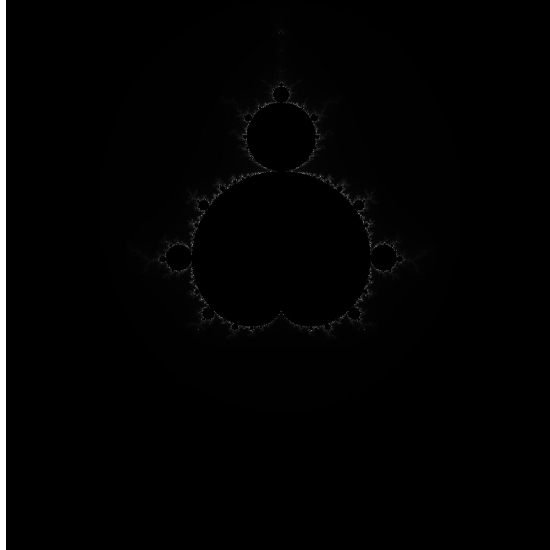
1.1 a)

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 1: Es besteht eine Race-Condition auf dem Array a über Index i, bei ungünstiger Ausführungsreihenfolge der Threads kann es dazu kommen, dass 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 2: Threads existieren 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 3: Hier ist die Variable x zunächst global definiert und wird somit implizit zwischen den Threads geteilt. 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 4: f ist global definiert und wird durch jeden Thread private gesetzt. Allerdings erhält jeder Thread eine uninitialisierte Kopie der Variable f, was bedeutet, dass das Initialisieren mit dem Wert 2 vor der Schleife nur für den Master-Thread sichtbar ist. Um diese Initialisierung auch für alle übrigen Threads sichtbar zu machen, ist es erforderlich, f mittels first-private 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 dieses Verhalten zu erreichen, muss x als last-private deklariert werden.
5. Example 5:

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 zugreifen 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, dass immer nur ein Thread gleichzeitig innerhalb des kritischen Abschnitts sein darf.

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

3.3 c)

CPU GPGPU FPGA MICs

4 Aufgabe 4

5 Aufgabe 5