Wissenschaftliches Rechnen III / CP III
Übungsblatt 4

Tizia Kaplan (545978)
Benjamin Dummer (532716)
Gruppe 10

25.05.2016

Online-Version: `https://www.github.com/BeDummer/CP3_UE4`

Aufgabe 4.i

Hier sollten die Laufzeiten für $n = 2^{24}$ Elemente für unterschiedliche Blockgrößen mithilfe von `nvprof ./a.out` gemessen werden. Vergleichend wurden die Simulationszeiten für die Implementierungen `reduceInterleaved` und `reducedUnrolling` aufgenommen. Gefunden wurde ein Minimum bei der Blockgröße 128 (siehe Abb. 1)

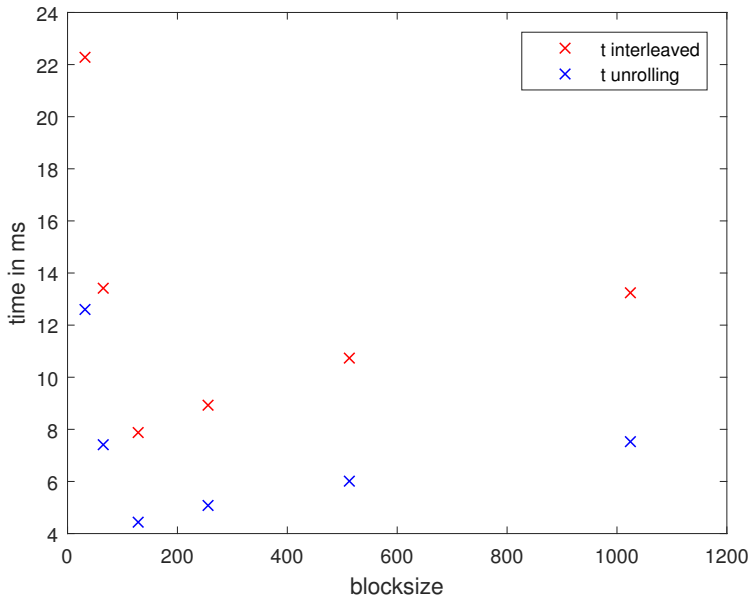

Abbildung 1: Blocksize vs. Simulationszeit für die unterschiedlichen Realisierungen der parallelen Reduktion (interleaved & unroll)

Aufgabe 4.ii

Nun konnten auch Global Memory Load Efficiency, Global Load Throughput und Achieved Occupancy gemessen werden. Hier wurden mittels `nvprof--metricsgld_efficiency,gld_throughput,achieved_occupancy./a.out` die Durchschnittswerte ermittelt. Die Ergebnisse sind in den Abb. 2 bis 4 gezeigt.

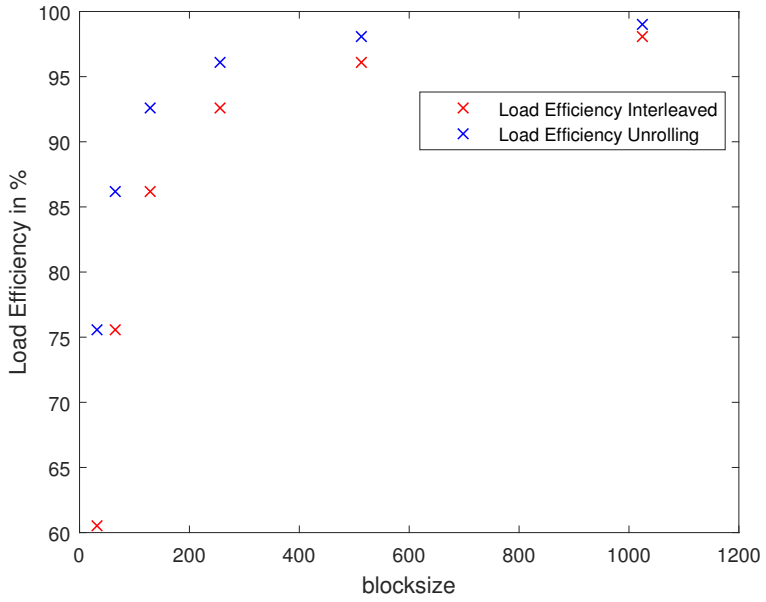

Abbildung 2: Blocksize vs. Global Memory Load Efficiency für die unterschiedlichen Realisierungen der parallelen Reduktion (interleaved & unroll)

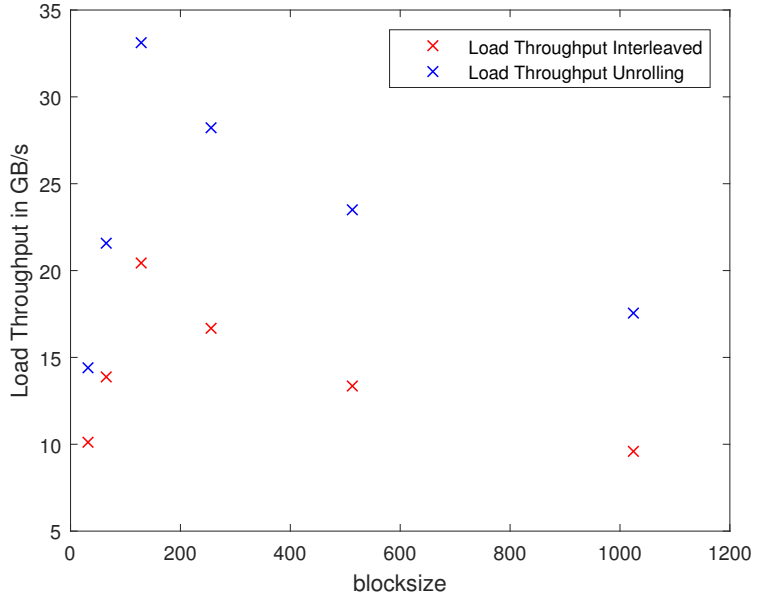

Abbildung 3: Blocksize vs. Global Load Throughput (Bandbreite) für die unterschiedlichen Realisierungen der parallelen Reduktion (interleaved & unroll)

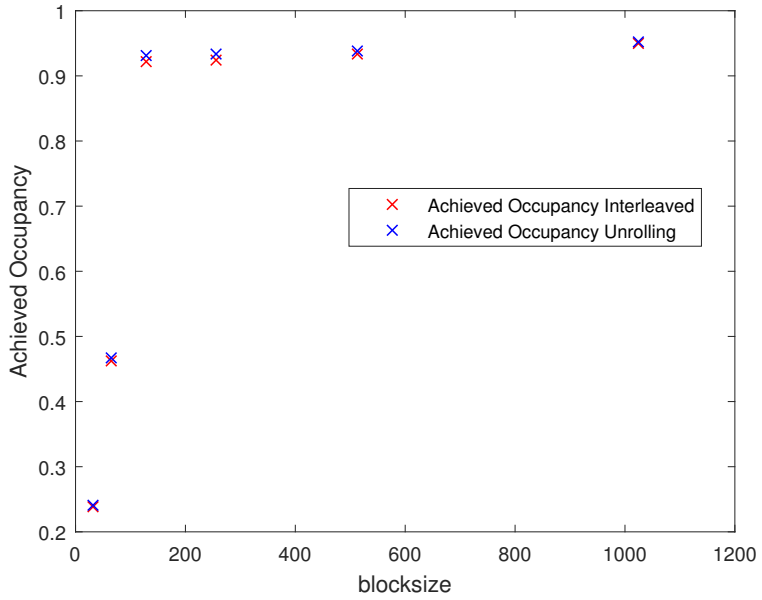

Abbildung 4: Blocksize vs. Achieved Occupancy für die unterschiedlichen Realisierungen der parallelen Reduktion (interleaved & unroll)

Aufgabe 4.iii

Man erkennt eine Laufzeitverbesserung für Blockgrößen bis 128 und diese erreicht dort ein Minimum. Die Verbesserung wird auch deutlich durch steigende Werte der in ii) gemessenen Parameter. Ab der Blockgröße 128 jedoch, ändern sich die *Load Efficiency* und die *Achieved Occupancy* nur unwesentlich - die Bandbreite sinkt jedoch wieder ab. Dementsprechend gibt es ein Minimum der Laufzeit bei Blockgröße 128.

In Abb. 1 lässt sich außerdem noch eine starke Differenz der Laufzeiten zwischen *Interleaved* und *Unrolling* erkennen. Unter einer Blockgröße von 128 ist die *Load Efficiency* ausschlaggebend für die Laufzeitunterschiede. Danach nimmt die Bandbreite den entscheidenden Einfluss auf die Differenz.

Eine Idee zur Erklärung des Minimums bei einer Blockgröße von 128 ist die Anzahl der Kerne pro *Streaming Multiprocessor*. Diese liegt bei 192. Wenn ein Thread-Block nun größer als 192 ist, können nicht mehr alle Operationen pro Block gleichzeitig ausgeführt werden und das System muss warten.

Aufgabe 4.iv

Hier wurde q als Potenz von 2 implementiert. Das Optimum der Laufzeit wurde für alle Blockgrößen (analog zu i)) im Bereich $1024 = 2^{10} \leq q \leq 2^{11} = 2048$ gefunden. Auffallend war, dass q einen Einfluss auf die absolute Laufzeit der unterschiedlichen Blockgrößen hatte: im Test wurde die schnellste Laufzeit von $t_{unrollOpt} = 1.42ms$ für eine Blockgröße von 1024 gefunden, allerdings variieren die Laufzeiten ab einer Blockgröße von 128 nur im Bereich von $10 - 20\mu s$.

Warum gilt das Laufzeitminimum für eine Blockgröße von 128 hier nicht?
Dadurch, dass mehrere Operationen pro Kern durchgeführt werden, wird die Bandbreite

optimal ausgeschöpft und der Einfluss des „Wartens “verschwindet.

Aufgabe 4.v

Ausgehend von unserer Analyse in iv) haben wir nun unsere optimierte Reduktion für den Datentyp `double` mit den optimalen Parametern (Blockgröße 1024 und $q = 2048$) erweitert. Der Vergleich zwischen den beiden Implementierungen (`int` vs. `double`) liefert folgende Ergebnisse:

Variablentyp	`int`	`double`
Laufzeit [ms]	1.47	2.01
Efficiency [%]	100	100
Throughput [GB/s]	47.8	69.4
Occupancy	0.9985	0.9992

Der Datentyp `double` hat einen höheren Speicherbedarf, dadurch kann die Bandbreite noch besser ausgenutzt werden. Da diese allerdings nach oben begrenzt ist, verlängert sich die Simulationszeit.

Anhänge

- Datei: `reduction_w_q.cu` (Suche nach optimalem q)
- Datei: `reduction_w_q_fix.cu` (`int`-Implementierung für Vergleich mit `double`-Implementierung)
- Datei: `reduction_double.cu` (`double`-Implementierung mit optimierten Parametern)