Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. Peter Sanders
Dennis Luxen, Vitaly Osipov,
Dennis Schieferdecker

Algorithm Engineering - Graphikprozessoren SS 2010
Übungsblatt 6

http://algo2.iti.kit.edu/cuda_10.php
{luxen|osipov|schieferdecker}@kit.edu
Ausgabe: Dienstag, 25.05.2010
Abgabe: Montag, 31.05.2010

Hinweis: Bei Problemen oder Fragen stehen wir jederzeit zur Verfügung

Aufgabe 6 (*Paralleler Quicksort*)

Auf dem letzten Übungsblatt wurde eine einfache Variante eines parallelen Quicksort Algorithmus' vorgestellt für den Spezialfall, dass alle Daten in den *shared memory* passen. Im folgenden wird dieser Ansatz erweitert, so dass er auf Folgen beliebiger Länge arbeiten kann, solange alle Daten in den globalen Speicher der Graphikkarte passen.
Zunächst führen wir ein Partitionierungsschema für Folgen im globalen Speicher ein. Der zugehörige *kernel* erhält als Eingabe den Zeiger auf eine Folge a der Länge n, die Grenzen $i,\ j$ des zu partitionierenden Bereichs sowie ein Pivotelement p. Die Ausgabe ist der partitionierte Bereich sowie die Grenzen der beiden Teilfolgen vor und nach dem Pivotelement. Das Partitionierungsschema unterteilt die gesamte Folge in Abschnitte zu je b Elementen, die jeweils von einem Block bearbeitet werden. Innerhalb eines Blocks ist ein Thread für genau eine Position der Folge verantwortlich.
Die Partitionierung besteht aus zwei Phasen. In der ersten Phase werden zwei Folgen L und G aus *interblock Offsets* bestimmt. $L_i,\ 1 \leq i \leq \lceil n/b \rceil$ ist gleich der Anzahl Elemente in Folge a, die kleiner dem Pivot sind und vor dem ersten Element von Block i der Folge stehen (Blöcke werden ab 1 nummeriert). Entsprechend für G_i und Elemente in a größer p. Außerdem wird die Gesamtzahl an Elementen, die kleiner bzw. größer dem Pivot sind (L_{sum}, G_{sum}), bestimmt.
In der zweiten Phase findet die Umordnung der Elemente von a im globalen Speicher statt. Dies geschieht blockweise entsprechend der parallelen Partitionierung aus dem letzten Übungsblatt. Einziger Unterschied ist, dass die verschobenen Elemente direkt in den globalen Speicher geschrieben werden und zu ihrem *intrablock Offset* aus l bzw. g noch der zugehörige *interblock Offset* aus L bzw. G addiert wird. Elemente, deren Index zwischen L_{sum} und $n - G_{sum}$ liegt, werden mit dem Wert des Pivotelements aufgefüllt.
Das Rahmenwerk des parallelen Quicksort wird im *host code* ausgeführt. Hier werden die noch zu partitionierenden Bereiche verwaltet und einzelne Partitionierungsvorgänge angestoßen. Die Verwaltung der Bereiche geschieht zweckmäßigerweise über eine Prioritätsliste, die größere Bereiche vorzieht. Ist ein Bereich klein genug, so dass er mit allen benötigten Hilfsdaten in den *shared memory* passt, kann der parallele Quicksort Algorithums aus dem letzten Übungsblatt angewendet werden.

a) Zur Bestimmung der *interblock Offsets* ermittelt jeder Block wieviele seiner Elemente kleiner und wieviele größer dem Pivot p sind. Diese Werte werden in zwei Hilfsfolgen zwischengespeichert, um anschließend ihre exklusive Präfixsumme zu berechnen und in L

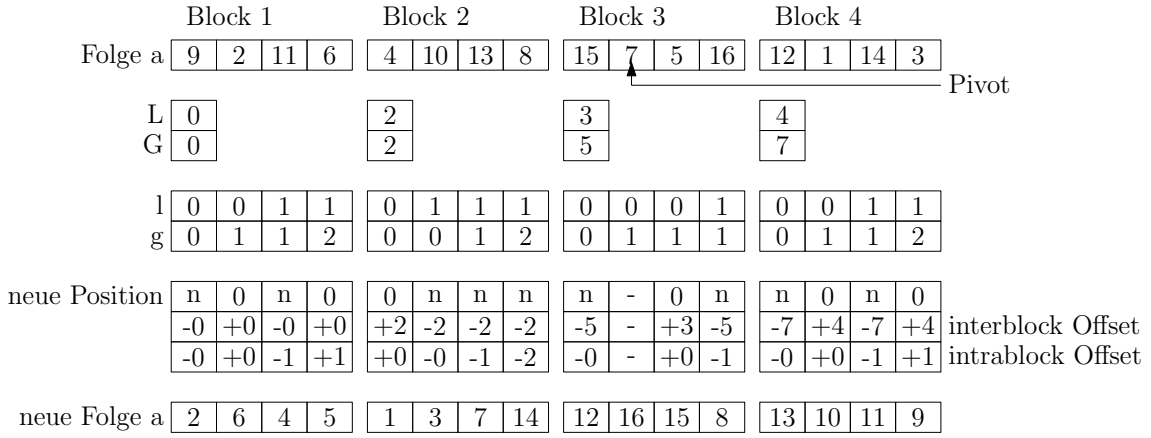

Abbildung 1: Die erste Zeile zeigt die Eingabefolge a aufgeteilt in 4 Blöcke mit $b = 4$. Die zweite und dritte Zeile geben die *interblock Offsets* L und G der einzelnen Blöcke an. Entsprechend für die *intrablock Offsets* l und g für jeden Block in Zeile 4 und 5. Anschließend ist dargestellt, wie sich die neuen Positionen der Elemente aus den Offsets berechnen. Zuunterst ist die partionierte Folge zu sehen. Die noch freien Stellen wurden mit dem Pivotelement aufgefüllt.

bzw. G abzulegen. Im selben Schritt werden die jeweiligen Gesamtsummen L_{sum} und G_{sum} berechnet. Ein Block bestimmt wieviele seiner Elemente kleiner dem Pivot sind, indem jeder Thread des Blocks in eine Hilfsfolge eine 0 oder 1 einträgt und anschließend eine parallele Reduktion auf der Folge durchgeführt wird. Entsprechend für Elemente größer dem Pivot.

Implementieren Sie den parallelen Partitionierer für Folgen im globalen Speicher. Wählen Sie eine geeignete Blockgröße, so dass die benötigten Daten zur Verarbeitung von mindestens einem Block in den *shared memory* passen. Verifizieren Sie die Korrektheit Ihrer Implementierung auf einer Zufallsfolge aus 32bit Ganzzahlen mit geeigneter Länge und einem zufällig aus dieser Folge gewählten Pivotelement. Testen Sie auch, ob Ihr Algorithmus korrekt funktioniert, wenn nur eine Teilfolge partitioniert werden soll.

Beachten Sie, dass das Ergebnis einer Partitionierung nicht direkt auf den gleichen Speicherbereich zurückgeschrieben werden kann. Verwenden Sie daher seinen zusätzlichen Puffer $\tilde{a}$, um die partitionierten Folge zu speichern.

b) Implementieren Sie nun das Rahmenwerk des parallelen Quicksort Algorithmus'. Da das Ergebnis einer Partitionierung nicht direkt in den gleichen Speicherbereich zurückgeschrieben werden kann, müssen Sie beim Zurückschreiben zwischen den Speicherbereichen von a und $\tilde{a}$ alternieren. Hierbei müssen Sie sich für alle Elemente merken, welcher der beiden Speicherbereiche a bzw. $\tilde{a}$ die jeweils aktuellen Daten hält.

Testen Sie Ihren Algorithmus, indem Sie eine Folge der Länge $n = 2^{24}$ aus zufälligen 32bit Ganzzahlen erzeugen und sortieren. Messen die Laufzeit Ihrer Implementierung für mehrere Durchläufe mit unterschiedlichen Zufallsfolgen. Überprüfen Sie auch die Korrektheit Ihres Algorithmus' für einen Durchlauf.

c) *(Zusatz–freiwillig)* Modifizieren Sie Ihren Algorithmus, so dass jeder Thread für k aufeinanderfolgende Elemente der Folge a verantwortlich ist. Testen Sie Ihren Algorithmus wie zuvor für verschiedene Werte von k (1, 2, 4, ...).

Lernziele – Woche 6

Nach Bearbeitung des vorliegenden Übungsblattes sollten Sie mit folgenden Themen vertraut sein:

- Parallelisierung von Quicksort