Karlsruher Institut für Technologie
Institut für Theoretische Informatik

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

Algorithm Engineering - Graphikprozessoren SS 2010
Übungsblatt 5

http://algo2.iti.kit.edu/cuda_10.php
{luxen|osipov|schieferdecker}@kit.edu
Ausgabe: Dienstag, 18.05.2010
Abgabe: Montag, 24.05.2010

Hinweis: Bei Problemen oder Fragen stehen wir jederzeit zur Verfügung

Aufgabe 1 (*Bitonisches Sortiernetzwerk*)

Ein *Sortiernetzwerk* ist ein spezieller Sortieralgorithmus, bei dem die Reihenfolge der Vergleiche nicht von der Eingabe abhängt. Es ist daher besonders gut für eine Implementierung in Hardware oder auf Graphikkarten geeignet.
Eine *bitonische Folge* ganzer Zahlen $a_{1..n}$ hat die Eigenschaft, höchstens zweimal ihre Monotonierichtung zu wechseln. Beispiel: $\{1,2,3,1,2,3,4\}$ ist bitonisch, $\{1,2,3,1,2,3,1\}$ nicht.
Die Grundoperation eines bitonischen Sortiernetzwerkes ist der *bitonische Vergleicher* B_n° mit $\circ$ eine der beiden Vergleichsoperatoren $\leq, \geq$. Der Vergleicher erhält als Eingabe eine Folge a der Länge n, mit n gerade und tauscht die Elemente a_i und $a_{i+n/2}, i \leq n/2$, falls $a_i \circ a_{i+n/2}$ nicht erfüllt ist. Für eine bitonische Eingabefolge der Länge n liefert $B_n^\leq$ als Ausgabe eine Folge, die sich in zwei bitonische Teilfolgen der Länge $n/2$ aufteilen lässt. In diesem Fall gilt außerdem, dass alle Elemente der ersten Teilfolge kleiner gleich allen Elementen der zweiten Teilfolge sind. Für $B_n^\geq$ gilt entsprechendes. Beispiel: $B_8^\leq(\{1,3,5,7,6,4,2,1\}) = \{1,3,2,1,6,4,5,7\}$.
Ein *bitonischer Mischer* sortiert eine bitonische Eingabefolge der Länge n, n Zweierpotenz. Er führt zunächst B_n° auf der gesamten Liste aus und wendet anschließend auf beiden Teilfolgen der Länge $n/2$ rekursiv den bitonischen Mischer an bis zum Basisfall $n = 1$. Aufgrund der oben genannten Eigenschaft von B_n° ist das Endergebnis eine sortierte Folge.
Für den Fall einer beliebigen Eingabefolge muss diese zunächst in eine bitonische Folge gewandelt werden. Dies erledigt der *bitonische Sortierer* durch rekursiven Aufruf von sich selbst. Er sortiert eine beliebige Folge der Länge n, n Zweierpotenz, indem er beide Teilfolgen der Länge $n/2$ rekursiv bis zum Basisfall $n = 1$ sortiert. Die zweite Teilfolge wird hierbei in umgekehrter Reihenfolge sortiert, um insgesamt eine bitonische Folge zu erhalten. Anschließend wird ein bitonischer Mischer auf die gesamte (bitonische) Folge angewendet.
Für den Fall, dass n keine Zweierpotenz ist, kann die Folge mit Nullen auf eine entsprechende Länge gebracht werden. Schematische Darstellungen beider Algorithmen finden Sie auf der nächsten Seite. Beachten Sie auch, dass das bitonische Sortiernetzwerk *in place* arbeitet.

a) *Theorie.* Zeigen Sie, dass die oben genannten Eigenschaften des bitonischen Vergleichers nur für bitonische Eingabefolgen erfüllt sind.

b) Implementieren Sie ein bitonisches Sortiernetzwerk. Beschränken Sie sich hierbei auf die Sortierung einer Folge im *shared memory* der Länge b. Da rekursiven Kernelaufrufe (bisher) nicht unterstützt werden, wandeln Sie die benötigten Rekursionen einfach in Schlei-

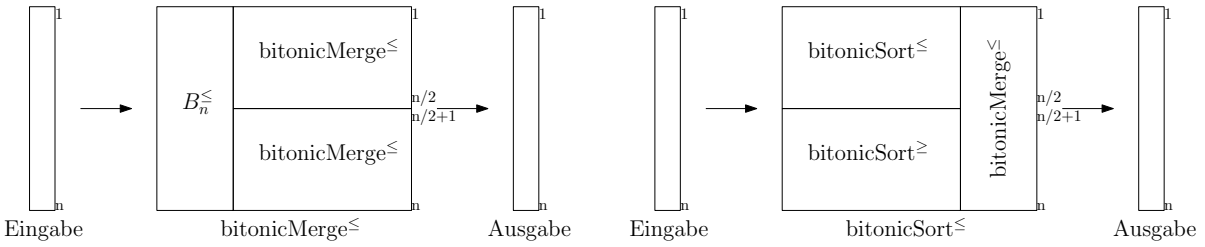

Abbildung 1: Schematische Darstellungen des *bitonischen Mischers* (links) und des *bitonischen Sortierers* (rechts) für den Fall der aufsteigenden Sortierung einer Eingabefolge der Länge n. Die Horizontale entspricht der Zeitachse und die Vertikale der Länge der betrachteten Folge.

fen um. Ist b bekannt, können die Schleifen auch ausgerollt werden, um einen kleinen Geschwindigkeitsvorteil zu erzielen.

Zeichnen Sie sich für ein besseres Verständnis zunächst für einen einfachen Fall, z.B. $n = 8$ oder $n = 16$, auf, welche Elemente wann miteinander verglichen (und ggf. getauscht) werden. Alle Vergleiche, die B_n° ausführt, finden parallel statt.

Jeder Thread Ihrer Implementierung sollte für eine (einfachere Variante) oder mehrere feste Positionen der Folge verantwortlich sein. Für den letzteren Fall bietet sich eine Zweierpotenz aufeinanderfolgender Elemente an. Achten Sie darauf an geeigneten Stellen eine Threadsynchronisierung einzufügen.

Geeignete Werte für b sind z.B. $128,\ 256,\ \ldots$

c) Testen Sie Ihren Algorithmus auf Korrektheit. Erstellen Sie dazu eine Folge von $n = 2^{24}$ zufälligen 32bit Ganzzahlen und sortieren Sie jeweils Teilfolgen der Länge b mit dem bitonischen Sortiernetzwerk. Wie lange benötigt Ihr Algorithmus um alle Teilfolgen zu sortieren?

Aufgabe 2 (*Paralleler Quicksort*)

Der *Quicksort Algorithmus* ist eines der bekanntesten und am häufigsten eingesetzten Sortierverfahren. Daher sollte Ihnen der serielle Algorithmus bereits bekannt sein. Falls Sie mit seiner Funktionsweise nicht mehr vertraut sind, schlagen Sie ihn bitte nach.
Prinzipiell besteht Quicksort aus einer Aneinanderreihung von Partitionierungsschritten auf passend gewählten Bereichen der zu sortierenden Folge a. Ein einzelner *Partitionierungsschritt* erhält als Eingabe die Folge a sowie den Start- und Endindex $1 \leq i,\ j \leq |a|$ der zu partitionierenden Teilfolge. Er wählt sich ein *Pivotelement* a_p mit $i \leq p \leq j$ und schiebt alle Elemente aus der Teilfolge, die kleiner als a_p sind vor das Pivotelement und alle anderen dahinter.
Der Quicksort Algorithmus wendet die Partitionierung zunächst auf die gesamte Folge an und anschließend rekursiv auf die Teilfolgen vor bzw. hinter dem Pivotelement. Eine parallele Implementierung parallelisiert genau den Partitionierungsschritt. Die noch ausstehenden Partitionierungen werden über eine geeignete Datenstruktur verwaltet, aus der immer eine zur Ausführung ausgewählt wird.
Im folgenden wird eine parallele Partitionierung vorgestellt für den Fall, dass Folge a sowie alle benötigten Hilfsdaten in den *shared memory* passen. Auf dem nächsten Übungsblatt wird der Algorithmus erweitert, so dass er auf größeren Folgen im globalen Speicher arbeiten kann.
Partitionierung:
Eingabe: Folge a der Länge b sowie Start- und Endindex i, j des zu paritionierenden Bereichs. Außerdem werden zwei Folgen l und g der Länge $j - i + 1$ benötigt, die auf 0 initialisiert werden.

Thread k ist verantwortlich für Element a_k. Threads außerhalb von $\{i, \ldots, j\}$ sind inaktiv.
Zu Beginn wählt Thread i ein Pivotelement a_p. Anschließend prüft jeder Thread k, ob $a_k < a_p$ oder $a_k > a_p$. Im ersten Fall setzt er $l_{k-i} = 1$, im zweiten Fall $g_{k-i} = 1$. Danach berechnen die Threads *in place* die exklusiven Präfixsummen der Folgen l und g sowie die zugehörigen Gesamtsummen l_{sum}, g_{sum}. Diese geben die neuen Positionen für jedes Element an. Für $a_k < a_p$ ist $i + l_{k-i}$ die neue Position von Element a_k. Für $a_k > a_p$ ist $j - g_{k-i}$ die neue Position von Element a_k. Die Elemente a_k mit $i+l_{sum} \leq k \geq j-g_{sum}$ werden mit dem Wert des Pivotelements beschrieben.
Anschließend werden $(i, i + l_{sum} - 1)$ und $(j - g_{sum} + 1, j)$ als noch zu verarbeitende Partitionierungen gespeichert, außer es handelt sich um triviale Partitionierungen der Länge 1.

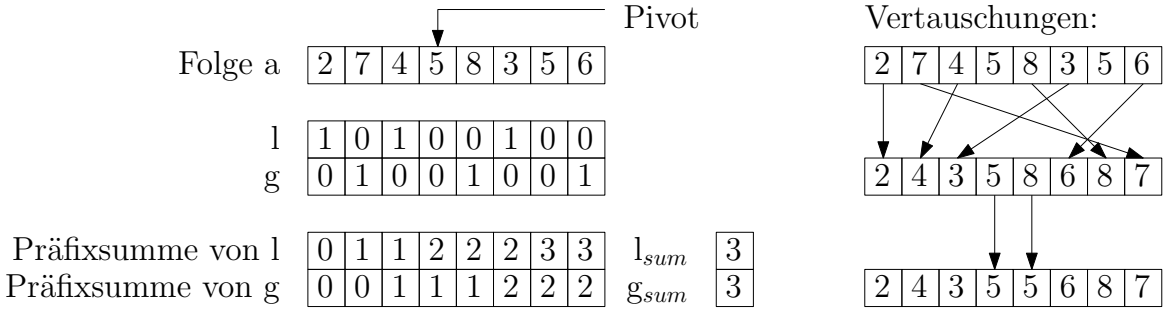

Abbildung 2: Schematische Darstellung eines Partitionierungsschrittes. Links ist die Eingabefolge a zu sehen und die Ergebnisse der Folgen l und g. Rechts sind zunächst die Vertauschungen aufgrund von l und g zu sehen und anschließend das Auffüllen mit Pivotelementen.

a) Implementieren sie den beschriebenen parallelen Quicksort Algorithmus. Wählen Sie b so, dass alle benötigten Daten in den *shared memory* passen. Wie groß kann b maximal gewählt werden bezogen auf die Graphikkarte in i10pc112? Beachten Sie insbesondere, genug Speicherplatz zum Sichern der noch auszuführenden Partitionierungen bereit zu stellen. Wieviele offene Partitionierungsschritte gibt es maximal in Abhängigkeit von b?

b) Bestimmen Sie die Laufzeit Ihrer Implementierung, indem Sie eine Folge aus $n = 2^{24}$ zufälligen 32bit Ganzzahlen erzeugen und jeweils Teilfolgen der Länge b sortieren. Überprüfen Sie auch die Korrektheit Ihres Sortierers.

c) (*) Erweitern Sie Ihren Quicksort Algorithmus, so dass jeder Thread für 4 Elemente der Folge verantwortlich ist. Testen Sie ihre Änderungen wie in Teilaufgabe b).

d) (*) Verwenden Sie nun Ihr Sortiernetzwerk aus Aufgabe 1 zum Lösen von Teilproblemen bestehend aus maximal 32 Elementen und testen Sie den geänderten Algorithmus erneut.

Aufgrund des Pfingstwochenendes dürfen die mit (*) gekennzeichneten Aufgaben auch erst zusammen mit Übungsblatt 6 abgegeben werden.

Lernziele – Woche 5

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

- Bitonisches Sortiernetzwerk
- Parallelisierung von Quicksort im *shared memory*