PARALLEL PROGRAMMING

Assignment 3

Scan

Tammo Johannes Herbert (319391), Rico Jasper (319396), Erik Rudisch (343930)

29.05.2013

Exercise 1 – Parallel Scan

(a) Data Parallel Scan

Gegeben sei ein Vektor $\vec{x}$ der Länge n. Im Falle eines sequentiellen Scans werden $n-1$ Operationen benötigt die ebenfalls in $n-1$ Schritten berechnet werden können. Der parallelisierte Scan benötigt dazu allerdings nur $m = \log_2 n$ Schritte. Der Schritt $j \in \{1, \dots, m\}$ wendet hier $n - 2^{j-1} + 1$ Operationen an. Insgesamt werden also $\sum_{j=1}^{m}\left(n - 2^{j-1} + 1\right) = (m-1)(n+1) + 2 \approx n \log_2 n$ Operationen ausgeführt; ein Vielfaches der Anzahl der sequentiellen Variante.

Die Ausführungsgeschwindigkeit hängt von der Anzahl p der Prozessoren ab. Ist $p \geq n$ so kann der Scan auch in m Zeitschritten berechnet werden, wobei viele Prozessoren zeitweise untätig sind. Sind weniger Prozessoren als Elemente vorhanden (d.h. $p < n$), so kann ein Prozessor entsprechend mehrere Elemente sequentiell berechnen. Das geht so lang, bis ein Schritt q mit $p = n - 2^{q-1} + 1$ erreicht ist. Ab hier gibt es dann wieder untätige Prozessoren. Die Schritte zu zuvor benötigen immer je so viele Zeitschritte, wie ein Prozessor Operationen ausführen muss. Daraus resultiert folgende Formel, die wir mit der Annahme das n und p ausreichend groß sind, abschätzen:

$$\sum_{j=1}^{m}\left\lceil\frac{n - 2^{j-1} + 1}{p}\right\rceil \approx \sum_{j=1}^{q}\frac{n - 2^{j-1} + 1}{p} + (m-q) = \frac{q(n+1) + 2^q - 1}{p} + (m-q)$$
$$\approx \left(\frac{n}{p} - 1\right) \cdot (\log_2(n - p + 1) - 1) + \log_2 n$$

Wobei $q = \log_2(n-p+1) + 1 \approx \log_2(n-p+1)$.

(b) Handling large numbers of elements

Spaltet man das Eingabearray in so viele Teilstücke wie Prozessoren auf, so können diese sequentiell berechnet werden. Jeder Prozessor bildet so die kumulierte Summe ihres Teilstücks. Die jeweils letzten Werte der Teilstücke werden nun vom primitiven Scan kumuliert. Die kumulierten Werte sind nun die Offsets die jeder Prozessor auf sein Teilergebnis hinzuaddiert. Der erste Prozessor addiert allerdings nichts, alle übrigen verwenden den kumulierten Wert vom Vorgänger.

Die serielle Scanphase benötigt $n - p$ Operationen die in $\frac{n}{p} - 1$ Zeitschritte ausgeführt werden können. Der primitive Scan führt dann $(\log_2 p - 1) \cdot (p+1) + 2$ Operationen in $\log_2 p$ Zeitschritten aus. Das aufaddieren beansprucht $n - \frac{n}{p}$ Operationen und ist nach $\frac{n}{p}$ Zeitschritten abgeschlossen. Insgesamt sind also $2n - \frac{n}{p} - p + (\log_2 p - 1)(p+1) + 2$ Operationen nötig. Wenn n und p ausreichend groß sind, kann man dies auch auf $2n - \frac{n}{p} - p + p \log_2 p$ schätzen. Dies geschieht in insgesamt $2\frac{n}{p} + \log_2 p - 1$ Zeitschritten.

Für die die erste Variante aus (a) sind $n \log_2 n$ Operationen notwendig währenddessen diese hier $2n - \frac{n}{p} - p + p \log_2 p$. Wenn n groß genug und p klein genug sind, so benötigt die zweite Variante also weniger Schritte da $2n - \frac{n}{p} - p$ nur linear ist und $p \log_2 p < n \log_2 n$. Dies lässt sich dadurch erklären, dass die zweite Variante eine Mischung aus einem reinem sequentiellen und einem reinen parallelen Scan ist. Ähnliches gilt auch für die Berechnungsdauer: Der Term $\left(\frac{n}{p} - 1\right) \cdot (\log_2(n - p +$

$1) - 1)$ aus (a) ist für große n und ausreichend kleine p größer als $2\frac{n}{p}$, genauso wie $\log_2 n > \log_2 p - 1$.

(c) Recursion

Ein rekursiver Scan könnte nun nach dem Teile-und-herrsche-Prinzip ein Array nun halbieren und diese rekursiv an sich selbst übergeben. Das letzte Element des ersten Arrays kann dann auf alle Elemente des zweiten addiert werden. Die Rekursion bricht bei Arrays der Größe zwei ab und gibt dieses und addiert auf das zweite Element das erste.

Ein Scan besteht also immer aus Komponenten: zwei Scan-Operationen auf halbierte Arrays durchführen und dann alle Elemente des zweiten Scans reparieren. Ist das Eingabe-Array n Elemente lang, so müssen $\frac{n}{2}$ Reparaturen durchgeführt werden. Die beiden rekursiv aufgerufenen Scans mussten auch entsprechend $2\frac{n}{4} = \frac{n}{2}$ Reparaturen ausführen. Dies setzt sich bis zur Abbruchbedingung fort, die in $\log_2 n$ Schritten erreicht ist. Demnach wurden $\frac{n}{2}\log_2 n$ Operationen durchgeführt. Es werden also nur halb so viele Operationen ausgeführt wie in (a). Gibt es mindestens so viele Recheneinheiten wie $\frac{n}{2}$ Elemente, so sind ebenfalls nur $\log_2 n$ Zeitschritte notwendig. Außerdem hätte dann zu jedem Zeitpunkt jeder Prozessor etwas zu tun.

Exercise 2 – Segmented Scan

Unser Operator ist folgendermaßen definiert:

$$\begin{pmatrix} x \\ 0 \end{pmatrix} \oplus \begin{pmatrix} y \\ 0 \end{pmatrix} = \begin{pmatrix} x+y \\ 0 \end{pmatrix}$$
$$\begin{pmatrix} x \\ 1 \end{pmatrix} \oplus \begin{pmatrix} y \\ 0 \end{pmatrix} = \begin{pmatrix} x+y \\ 1 \end{pmatrix}$$
$$\begin{pmatrix} x \\ 0 \end{pmatrix} \oplus \begin{pmatrix} y \\ 1 \end{pmatrix} = \begin{pmatrix} y \\ 1 \end{pmatrix}$$
$$\begin{pmatrix} x \\ 1 \end{pmatrix} \oplus \begin{pmatrix} y \\ 1 \end{pmatrix} = \begin{pmatrix} y \\ 1 \end{pmatrix}$$

Exercise 3 – Implementation

- Auf verteilten Systemen ist Variante (b) zu bevorzugen. Alle Knoten berechnen lokal die kumulierte Summe zunächst auf serielle Weise. Mit einem primitiven Scan werden dann die Zwischenergebnisse verteilt. Da der primitive Scan auf genau so vielen Elementen wie Knoten arbeitet, ist dies ideal. Danach kompensiert jeder Knoten für sich den Offset.
- Für Systeme mit gemeinsamen Adressraum schlagen wir Variante (a) vor, da es hier notwendig ist auf demselben Speicher zu arbeiten. Der Adressabstand zwischen zwei zu addierenden Elementen nimmt nach jedem Schleifendurchlauf zu und macht deshalb geteilten Speicher notwendig. Allerdings ist hier auch (c) möglich.
- Variante (c) ist auf Grund der Natur des Teile-und-Herrsche-Prinzips in gewisser Hinsicht für die hierarchische Struktur des Speichers von GPU-Architekturen geeignet, solange jede Rekursionsstufe auch auf einen Speicherlevel zugeordnet werden kann. Ansonsten ist auch eine Variante mit einer festen Rekursionstiefe möglich. Hat man die größtmögliche Tiefe erreicht, kann man wie in (b) auf einen sequentiellen Scan wechseln.