Präfix Summen

Für eine Folge von Zahlen $x_0, x_1, ..., x_n$ ist die Folge der Präfixsummen (exklusiv) definiert als $0, x_0, x_0 + x_1, ..., \sum_{i=0}^{n-1} x_i$ (Summe aller vorherigen Elemente der Folge).

Beispiel:

Folge	3	2	1	2	1	4	3	2	4	3
PräfixSummen	0	3	5	6	8	9	13	16	18	22

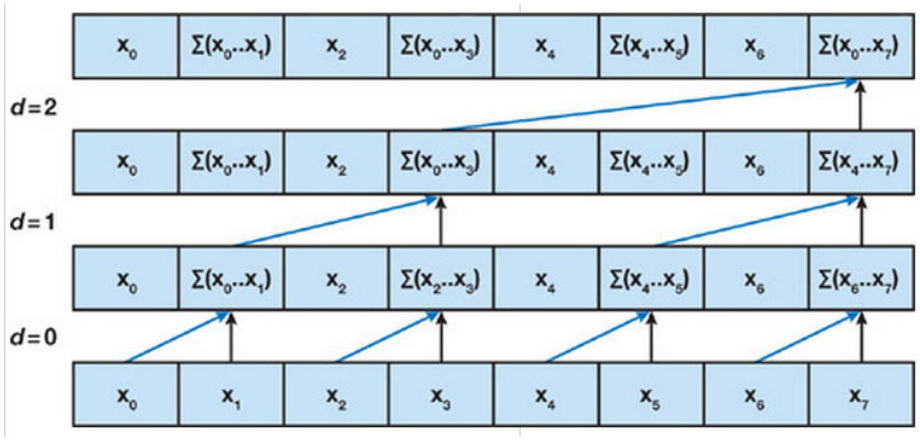
Iterativ lässt sich die Folge der Präfix-Summen auf naheliegende Weise berechnen:

```
void calcPrefixSums(int feld[], int prefixsum[], int size) {
   prefixsum[0] = 0;
   for (int i=1 ; i<size ; i++)
      prefixsum[i] = prefixsum[i-1]+feld[i-1];
}
```

Die Parallelisierung der Präfis-Summen-Berechnung ist ziemlich tricky:

Die Berechnung erfolgt in 2 Phasen, einer sog. Up-Sweep- und einer Down-Sweep-Phase. Wir gehen dabei davon aus, dass die Anzahl der Feldelemente n eine 2er-Potenz ist: $n = 2^k$

In der Up-Sweep-Phase werden wie in der folgenden Skizze illustriert, von unten nach oben Summen berechnet (die Ergebnisse werden jeweils im Original-Feld gespeichert, die einzelnen Stufen sind nur zur Illustration separat skizziert). Dabei wird von unten nach oben ein Binärbaum aufgebaut, dessen Knoten aber nicht gespeichert werden.



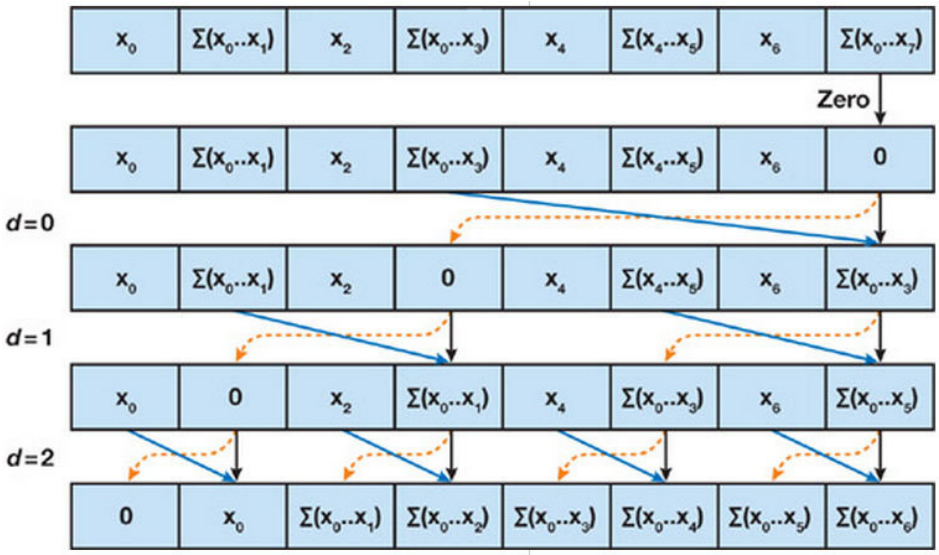
Quelle: GPU-Gems 3

In Pseudocode sieht die Up-Sweep-Phase etwa so aus:

```
for (int d=0 ; d<k ; d++)
  for (int i=2^(d+1)-1 ; i<n ; i+=2^(d+1)) // diese Schleife kann parallelisiert werden
     x[i]=x[i]+x[i-2^d];                   // Bemerkung: 2^a = 1<<a
```

Präfix Summen

In der Down-Sweep-Phase wird der Binärbaum nun wie in der folgenden Skizze illustriert von oben nach unten traversiert, um die Präfix-Summen zu bilden. Dabei wird zunächst der Wurzelknoten auf 0 gesetzt, dann wird der Baum von oben nach unten (beginnend mit dem Wurzelknoten) traversiert, wobei an jedem Knoten jeweils der linke Kindknoten auf den Wert des aktuellen Knotens und der rechte Kindknoten auf die Summe der Werte des aktuellen Knotens und des früheren Wertes des linken Kindknotens gesetzt wird.



Quelle: GPU-Gems 3

In Pseudocode sieht die Down-Sweep-Phase etwa so aus:

```
x[n-1]=0;
for (int d=k-1 ; d>=0 ; d--)
  for (int i=2^(d+1)-1 ; i<n ; i+=2^(d+1)) { // diese Schleife kann parallelisiert werden
     tmp = x[i];
     x[i] += x[i-2^d];
     x[i-2^d] = tmp;
  }
```

Die Datei `PräfixSummen_parallelisierbar.cpp` enthält ein lauffägies Programm, das die Präfix-Summen mit Hilfe dieses Algorithmus iterativ auf der CPU berechnet.

Präfix Summen

Aufgabe:

Implementieren Sie diesen Algorithmus in OpenCl auf der GPU, indem Sie in der Up-Sweep- und der Down-Sweep-Phase jeweils die mit dem entsprechenden Kommentar versehene Schleife parallelisieren. Dabi ist folgenden zu tun bzw. zu beachten:

- Implementieren Sie den Algorithmus für ein Feld mit fester Länge 256
- Benutzen Sie dazu 1 Workgroup mit 256 Workitems (das Programm läuft damit nur auf einer Compute Unit der GPU. Später kann der jetzt implementierte Teil dazu benutz werden, ein größeres Feld in 256-Element-Stücke aufzuteilen, die Stücke separat zu berechnen und anschließend wieder zusammenzusetzen)
- Als Ausgangspunkt können Sie das „HelloWorldCL“-Beispiel verwenden, das auf transfer (Laufwerk T) zu finden ist.
- Im cpp-File müssen Sie:
  - für das Ursprungsfeld und für das Ergebnisfeld jeweils einen Buffer erzeugen (`clCreateBuffer`). Die Daten des Urspungsfeldes müssen in den entsprechenden Buffer geschrieben werden.
  - die global_work_size und die local_work_size korrekt setzen
  - Ihren Kernel aufrufen
  - Die Ergebnisdaten aus dem Buffer lesen
- Gehen Sie bei der Implementierung des Kernels wie folgt vor:
  - Kopieren Sie zunächst die Eingabedaten ins local-Memory. Führen Sie die Berechnungen im local-Memory durch, und kopieren Sie dann das Ergebnis ins Ergebnis-Feld im global-Memory.
  - Beachten Sie, dass nach bzw. vor den Kopiervorgängen sowie bei jedem Schleifendurchgang der äußeren Schleifen der Up-Sweep- und Down-Sweep-Phasen eine Synchronisation nötig ist.