Projekt „Count Sort“ für den 11.05., 18.05. und 25.05.2018

Algorithmus:

Der Countsort Algorithmus arbeitet nach den folgenden Schritten:

Startend von einem Array mit Zahlen von 1 bis N (hier N = 7):

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
2	5	3	0	2	3	0	3

(1) Erzeugt ein Histogramm, welches die Häufigkeit jeder Zahl im Array beschreibt:

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
2	0	2	3	0	1

(2) Berechnet für jedes Element e im Array C die Anzahl von Einträgen im Array A welche kleiner/gleich dem Element e sind (prefix sum um 1 verschoben):

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
0	2	2	4	7	7

(3) Übertragt sukzessive die Werte aus A in einem sortierten Vektor B und zwar genau an der Stelle im Zielvektor, die der Hilfsvektor C für die entsprechende Zahl angibt. Vor der Schleife ist dies immer die erste Stelle, an der das Element e auftauchen wird. Nach dem Übertragen jedes Elements wird zusätzlich der Wert in C[e] inkrementiert. Das nächste gleiche Element wird deswegen eine Stelle weiter hinten im Zielvektor eingefügt.

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	C[0]	C[1]	C[2]	C[3]	C[4]	C[5]	B[0]	B[1]	B[2]	B[3]	B[4]	B[5]	B[6]	B[7]
2	5	3	0	2	3	0	3	0	2	2	4	7	7			2					
2	5	3	0	2	3	0	3	0	2	3	4	7	7			2					5
2	5	3	0	2	3	0	3	0	2	3	4	7	8			2		3			5
2	5	3	0	2	3	0	3	0	2	3	5	7	8	0		2		3			5
2	5	3	0	2	3	0	3	1	2	3	5	7	8	0		2	2	3			5
2	5	3	0	2	3	0	3	1	2	4	5	7	8	0		2	2	3	3		5
2	5	3	0	2	3	0	3	1	2	4	6	7	8	0	0	2	2	3	3		5
2	5	3	0	2	3	0	3	2	2	4	6	7	8	0	0	2	2	3	3	3	5

Anwendung:

Für die Übung sortieren wir eine Liste mit Eintragungen bestehend aus (Alter|Name) nach Alter. Der Wertebereich für das Alter ist 0-120 gespeichert als int, die Namenswerte sind char[32].

Aufgaben (1. Woche):

1. Schreiben Sie ein serielles C Programm "list_gen.c", das eine zufällige Liste aus Alter und Namen mit N Einträgen (als Kommandozeilenparameter) erzeugt und ausgibt. Als zweiter Parameter soll der Startwert für den random number generator S angegeben werden können (`srand(S)`). Zum Generieren zufälliger Namen verwenden Sie die bereitgestellte Funktion `gen_name(const char[32] buffer)` im Header `"people.h"`.

   ```
   Beispiel:
   [user@kreusspitze sort]$ ./list_gen 5 1337
    41 | Isela Stayer
    27 | Lanell Benberry
    61 | Chaya Kijek
    35 | Malika Harbour
    44 | Mitzi Mccalley
   ```

2. Implementieren Sie den countsort Algorithmus sequenziell am CPU um die generierte Liste zu sortieren. Das Programm "list_sort.c" soll die gleichen Parameter wie "list_gen" übernehmen, und die Liste erst unsortiert und dann sortiert ausgeben.

3. Überlegen Sie für jeden Schritt des Countsort Algorithmus, wie er parallelisiert werden kann. Beachten Sie dabei Faktoren wie Korrektheit, Synchronization und Performance auf GPUs sowie CPUs. Schreiben Sie Ihre Überlegungen in einer Datei "parallelization_plan.txt" nieder.

Ressourcen:

- people.zip: Enthält Header people.h sowie Textdateien zur Namensgenerierung.

Hinweise:

- Um eine gewisse Struktur zu erzielen, ist der Datentyp für Personen in people.h vorgegeben:
  ```
  typedef struct {
        int age;
        name_t name;
  } person_t;
  ```
  Diese Struktur ist auch zu verwenden!
- Gliedern Sie ihr Programm in logische Funktionen, z.B. für Aufgabe 1:
  `void generate_list(person_t** list, int entries) { ... }`
  `void print_list(person_t* list, int entries) { ... }`
  Eine Gliederung nach den Schritten des Algorithmus ist bei Aufgabe 2 sinnvoll.

Abgabe (1. Woche):

- list_gen.c, list_sort.c, parallelization_plan.txt für die jeweilige Aufgabe
- Per email an herbert.jordan@uibk.ac.at, 1 Abgabe pro Gruppe
  Betreff: “[PS703106] [UE07] GR_XX - NAME1 NAME2 NAME3”
  Vor (!) Übungsbeginn, am 11.05.2018

Aufgaben (2. Woche):

Implementieren Sie die Prefix Sum (2. Schritt des Algorithmus) parallel in OpenCL, wie in der Vorlesung besprochen.

1. Implementieren der Prefix Sum für eine einzelne work group nach Hillis und Steele. (`hillissteele.{c/cl}`)
2. Optimierte Implementierung für einzelne work groups. (`downsweep.{c/cl}`)
3. Erweiterung der Implementierung für work groups auf beliebig große Arrays. (`prefixglobal.{c/cl}`)

   Für jede der Varianten sollte ein Testprogramm bereitgestellt werden!

Abgabe (2. Woche):

- Programme `hillissteele/downsweep/prefixglobal.`
- Per email an herbert.jordan@uibk.ac.at, 1 Abgabe pro Gruppe
  Betreff: "[PS703106] [UE08] GR_XX - NAME1 NAME2 NAME3"
  Vor (!) Übungsbeginn, am 18.05.2018

Aufgaben (3. Woche):

Implementierung von list_sort aus Aufgabe 1 in OpenCL, unter Zuhilfenahme der Prefix Sum Implementierung aus Aufgabe 2.

1. Implementieren Sie die Häufigkeitsberechnung (1. Schritt des Algorithmus) parallel in OpenCL.
2. Implementieren Sie das übertragen der Werte (3. Schritt des Algorithmus) parallel in OpenCL.
3. Schreiben Sie ein Testprogramm "countsort_bench.c", das dieselben Parameter übernimmt wie die Programme aus der ersten Woche, die Listen aber nicht ausgibt, sondern stattdessen die Zeit um Sie sequenziell bzw. mit OpenCL zu sortieren.

   Beispiel:

```
[user@kreusspitze sort]$ ./countsort_bench 2000000 1337
Sequential:      12.11 ms
OCL Device 0:     4.32 ms
OCL Device 1:     2.45 ms
```

Abgabe (3. Woche):

- Programm `countsort_bench.c`
- Per email an herbert.jordan@uibk.ac.at, 1 Abgabe pro Gruppe
  Betreff: "[PS703106] [UE09] GR_XX - NAME1 NAME2 NAME3"
  Vor (!) Übungsbeginn, am 25.05.2018