Übungsblatt 9, für den 8.1.2026

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 dem Element e sind (prefix sum!):

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 int32_t, die Namenswerte sind char[32].

Aufgaben:

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:
   ```
   [petert@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. Implementieren Sie die Häufigkeitsberechnung (1. Schritt des Algorithmus) parallel in OpenCL.

4. Implementieren Sie den 2. Schritt unter Zuhilfenahme der existierenden Prefix Sum Implementierung.

5. Implementieren Sie das übertragen der Werte (3. Schritt des Algorithmus) parallel in OpenCL.

6. Schreiben Sie ein Testprogramm "countsort_bench.c", das die selben Parameter übernimmt, die Listen aber nicht ausgibt, sondern stattdessen die Zeit um Sie sequenziell bzw. mit OpenCL zu sortieren.
   Beispiel:
   ```
   [petert@kreusspitze sort]$ ./countsort_bench 2000000 1337
   Sequential:      12.11 ms
   OCL Device 0:     4.32 ms
   OCL Device 1:     2.45 ms
   ```

   Wieviele Personen kann Ihre Implementierung auf den **NV** und **AMD** nodes je maximal **in 250 ms** sortieren? (Zeitmessung rein auf GPU, erster Kernel Start bis letzter Kernel Ende)

Ressourcen:

- `ex9_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 {
    int32_t 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.

- Online-recherche ist absolut kein Problem, aber jede spezialisierte Optimierung muss im Detail erklärt werden können.
- Die Gruppen mit den **~2 schnellsten (= maximal sortierte Größe in 250ms) Lösungen erhalten je einen Bonuspunkt**. Es gibt auch Bonuspunkte für sinnvoll-kreative Lösungsansätze.
  *Alle Optimierungen sind für diesen Zweck grundsätzlich **erlaubt**, solange die Semantik des Programms korrekt erhalten bleibt.*

Zusätzliche Informationen / Klarstellungen:

- Beim (zufälligen) erzeugen der Liste pro Person als erstes den Namen und dann das Alter generieren.
- Der Algorithmus muss **nicht** notwendigerweise als stable sort implementiert werden.
- Wenn gewisse Teile sequenziell besser als parallel arbeiten ist das durchaus in Ordnung, sollte aber geprüft werden.
- **Der komplette Sortiervorgang muss aber ausnahmslos am GPU laufen.**

Abgabe:

- Per Email an peter.thoman@uibk.ac.at
  Betreff: "`[gpu-computing-2025] [UE9] NACHNAME1, NACHNAME2, NACHNAME3`"
  Vor (!) VU-Beginn
  1 Abgabe pro Gruppe – Gruppenmitglieder in der Email auflisten

  Format:
  Archiv (.tar, .tar.gz, .zip, …) mit einem Folder.
  Folder enthält source + makefile.
  Letzteres muss out of the box auf `ifi-cluster.uibk.ac.at` funktionieren.
  Messdaten, schriftliche Antworten etc. als .txt, .md, .pdf und/oder .csv.