Assignments 7, 8, and 9, due May 10th, 17th, and 24th 2019 Project "Count Sort"

Algorithm:

The count sort algorithm consists of three steps. Start with an array of unsorted integer numbers:

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

- Create a histogram of the occurrence of each number:

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

- For each element c in C, compute the number of elements in A which are less or equal to c (prefix sum shifted by 1):

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

- Copy each value a in A into a new array B. The target position in B is determined by the corresponding number for a in C. After each copy operation, the corresponding number in C will incremented, thus writing the next element of the same value in the adjacent position:

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

Use case:

- For these assignments, we sort a list of `(age|name)` entries according to age. Age is represented as an `int` with a value range from 0 to 120, name is represented as `char[32]`.

First assignment (until May 10th)

Assignment:

- Write a sequential C program `list_gen.c` that creates and prints a list of N random `(age|name)` entries, with N being a command line parameter. The seed for the random number generator should be a second command line parameter. For generating random names, use the provided function `gen_name(const char[32] buffer)` in header file `people.h`.

  Example:
  ```
  [philipp@hudson sort]$ ./list_gen 5 1337
  41 | Isela Stayer
  27 | Lanell Benberry
  61 | Chaya Kijek
  35 | Malika Harbour
  44 | Mitzi Mccalley
  ```
- Implement a sequential version of the count sort algorithm for sorting the generated list on the CPU. The program `list_sort.c` should take the same parameters as `list_gen.c` and print the list of names twice, first unsorted and then sorted.
- Come up with a parallelization strategy for every step of the count sort algorithm. Consider factors such as correctness, synchronization, and performance on GPUs and CPUs. Write down your parallelization strategy and considerations into a file `parallelization_plan.txt`.

Resources:

- `people.zip`
  Holds a header file `people.h` and text files for generating names.

Hints:

- In order to support a structured approach, the data type for the `(age|name)` entries is defined in `people.h`:
  ```
  typedef struct {
      int age;
      name_t name;
  } person_t;
  ```
  This data structure must be used for all exercises!
- Organize your program into appropriate functions, for example:
  ```
  void generate_list(person_t** list, int entries) { … }
  void print_list(person_t* list, int entries) { … }
  ```
  Also, a structure resembling the individual steps of the algorithm is required for the second assignment.

Solution upload:

- `list_gen.c`, `list_sort.c`, `parallelization_plan.txt`
- Via e-mail to philipp.gschwandtner@uibk.ac.at – one submission per group only!
  Subject: “`[PS703106] [AS07] GR_## - NAME1, NAME2, NAME3`”
  Solution must be submitted before Friday May 10th, 09:15!

Second assignment (until May 17th)

Assignment:

- Implement an OpenCL version of the prefix sum algorithm (second step of the count sort algorithm) as discussed in the lecture:
  - Prefix sum implementation for a single work group according to Hillis and Steele (`hillissteele.{c/cl}`).
  - Optimized implementation using down-sweep step (`downsweep.{c/cl}`).
  - Extension to arbitrarily large arrays using multiple work groups (`prefixglobal.{c/cl}`).
- For each of the aforementioned variants, a test program must be provided.

Solution upload:

- `hillisteele.{c/cl}`, `downsweep.{c/cl}`, `prefixglobal.{c/cl}`
- Via e-mail to philipp.gschwandtner@uibk.ac.at – one submission per group only!
  Subject: "`[PS703106] [AS08] GR_## - NAME1, NAME2, NAME3`"
  Solution must be submitted before Friday May 17th, 09:15!

Third assignment (until May 24th)

Assignment:

- Implement `list_sort` of the first assignment in OpenCL using the prefix sum implementation of the second assignment.
  - Implement the histogram step (first step of count sort) using OpenCL
  - Implement the element copying (third step of count sort) using OpenCL
  - Write a test program `countsort_bench.c`, which takes the same parameters as the program of the first assignment. However, instead of printing the names, it should just measure and print the time required for sorting.

    Example:
    ```
    [philipp@hudson sort]$ ./countsort_bench 2000000 1337
    Sequential: 12.11 ms
    OCL: 4.32 ms
    ```

Solution upload:

- `countsort_bench.c`, including all involved source/header files
- Via e-mail to philipp.gschwandtner@uibk.ac.at – one submission per group only!
  Subject: "`[PS703106] [AS09] GR_## - NAME1, NAME2, NAME3`"
  Solution must be submitted before Friday May 24th, 09:15!