CS 6023 GPU Programming
Assignment 3

Handout date: 14/09/2018
Due date: 23/09/2018 (end of day)

General instructions:

1. The goal of this assignment is to let you explore ways to improve the running time of a CUDA program.
2. This is an individual assignment. Collaborations and discussions with others are not encouraged. However, if you do collaborate with anyone, you are requested to explicitly state the same in your submitted report.
3. You have to use C with CUDA for your implementation.
4. You have to turn in three components: (a) well documented code (.cu files for each programming question), (b) output files (.txt for each relevant programming question), and (c) a detailed report (single .pdf file) of the results of the experiment. All these are to be submitted on Moodle. No hard copy is required.
5. Code will be checked for functional correctness, quality of code, and performance. Report will be verified for completeness and to-the-point answers to the subjective questions posed.
6. If you have doubts please raise them in the Moodle Q&A forum, and follow the same regularly. Also, to receive programming and other support, attend the TA session on Thursday 10am at the usual classroom CS 36.
7. Codes generating incorrect output will be awarded zero marks.
8. We provide you with code snippets for taking inputs. Please include these code snippets as it is in your source code. Modifying the code snippets may result in an incorrect output and lead to consequences mentioned in 7.

Submission instructions:

1. Make a report PDF as *<username>_report.pdf*, where <username> is the username used by you to login to the GPU cluster. For many it may be your roll number.
2. Put all files inside a folder named your username and compress the folder into a tarball in the following format : *username_A03.tar.gz*
3. Upload the tarball in Moodle. For instructions on how to do this contact the TAs.

Question (5 points):

Write a CUDA code to generate “N-count-grams” of a text, and accumulate the frequencies of each of the unique N-count-grams from the text, which can later be represented as a histogram.

N-count-grams of a text are our own created definition to mean sequences of lengths of contiguous words in a window of size N moving in a sentence. For example, let us consider the sentence *“The quick brown fox jumps over the lazy dog”*. For this, we have the following N-grams:

1-grams: “the”, “quick”, “brown”, “fox”, “jumps”, “over”, “the”, “lazy”, “dog”
2-grams: “the quick”, “quick brown”, “brown fox”, “fox jumps”, “jumps over”, and so on.
3-grams: “the quick brown”, “quick brown fox”, “brown fox jumps”, “fox jumps over”, and so on.
9-gram will be the whole sentence itself.

Corresponding to N-grams we have N-count-grams which provide the number of characters in the words of the N-grams. For example for the N-grams listed above the corresponding N-count-grams respectively are:

1-count-grams: 3, 5, 5, 3, 5, 4, 3, 4, 3
2-count-grams: 3 5, 5 5, 5 3, 3 5, 5 4
3-count-grams: 3 5 5, 5 5 3, 5 3 5, 3 5 4
9-count gram: 3 5 5 3 5 4 3 4 3

So, for a sentence with `M` words, there are a total of `[M - (N - 1)]` N-count-grams, including possible repetitions.

While, we are working with N-count-grams, N-grams of texts can be very useful to predict the next word in a sentence by calculating the probabilities of different words that can come after a particular sequence of words. You can learn more about it in the following wiki article: https://en.wikipedia.org/wiki/N-gram

This question will be evaluated in two parts:

(a) 2 marks: correctness of the code
(b) 3 marks: relative, based on the running time on Tesla K40 (compared to all submissions)

Your code should take input N and the filename of the text from the command line argument (as given in the instructions later). A text on Shakespeare’s work has been provided on the moodle (*shaks.txt*, 17123 words), on which your code will be tested for correctness. The output will contain all the N-count-grams with their respective count of occurrences printed onto *<username>_out.txt* file in the format mentioned later.

We will test your code and measure performance for N = 1, 2, 3, 4, and 5. The performance will be tested on another text file to discourage over-fitting, i.e., using methods that are specifically optimized for a given input file.

Submit the source code as <username>.cu
Mention the optimization methods you have applied to reduce the running time of the program in the report titled <username>_report.pdf

---

Input/Output Instructions:

Add the following at the start of the file:

```
#include <stdio.h>
#define MAXWORDS 20000
```

Use the following code snippet to take input:

```
int N = atoi(argv[1]);      // For calculating N-count-grams
char *filename = argv[2];   // Filename: shaks.txt
char words[MAXWORDS * 20]; // Stores all words in 1D array
                           // Single word length is bounded by 20
char curWord[40];          // Take input string into this
int totalWordCount = 0;    // Count of number of words read
FILE *ipf = fopen(filename, "r");
while (fscanf(ipf, "%s ", curWord) != EOF
           && totalWordCount < MAXWORDS) {
     checkWord(curWord, ...);    // Check for word properties
                                 // and update 'words[]' array.
                                 // Modify this section according
                                 // to below mentioned properties
     totalWordCount += 1;
}
fclose(ipf);
```

We define a *word* to have the following properties:

a) No punctuations at the beginning or end of the string. For e.g., if you read "*hello!*", it needs to be modified to "*hello*".
b) After doing step (a), any hyphens inside a string will make it multiple words. For e.g., "*never-resting*" gets changed to two words: *"never"* and *"resting*".
c) Apostrophe inside a string is considered as part of the word and is accounted for in its length. For e.g., "*would've*" is considered a single word of length 8.
d) Apostrophes at the end of a string (e.g., for plural words) are not considered as part of the word, as they do not satisfy (a). For e.g., "*Chairs' "* gets changed to "*Chairs*".
e) Any punctuation inside a string (i.e., not at the beginning or end) is considered as part of the word. Just hyphens are treated specially, as mentioned in (b).

Output format (say, for N = 2) in *<username>_out.txt*:

```
<word1_len_1> <word2_len_1> <count_1>
<word1_len_2> <word2_len_2> <count_2>
```

and so on, for all the N-count-grams. The order of printing does not matter.

As an example for the sentence *"The quick brown fox jumps over the lazy dog"*, and for N = 2 the output of the file could have
3 5 2
5 5 1
5 3 1
5 4 1

Note from the example above that the N-gram-count (3, 5) is different from (5, 3), i.e., the ordering of the numbers matters in defining distinct N-grams.