COMP5112 / MSBD5009 Parallel Programming (Spring 2022)

Assignment 3: CUDA Programming

Submission deadline: **23:59 (pm) on May 4, 2022**

I. General Notes

1. This assignment counts for 15 points.
2. This is an individual assignment. You can discuss with others and search online resources, but your submission must be your own code. Plagiarism checking will be enforced, and penalty will be given to all students involved in an incident of academic dishonesty.
3. Add your ***name, student id, and email*** at the first two lines of comments in your submission.
4. Submit your assignment through Canvas before the deadline.
5. All submission will be compiled and tested uniformly on given machines of the course (CSE Lab 2 machines for COMP5112 and Azure virtual machines for MSBD 5009).
6. Please direct any inquiries about this assignment to the designated TAs listed in the CANVAS assignment page.
7. No late submission will be accepted!

II. Problem Description

This assignment is on the CUDA parallelization of a ***super-mer*** generation algorithm given an input list of DNA fragments, called ***reads***. Each read is represented as a string of length n, and the characters in the string include 'A', 'T', 'C', and 'G'. A ***k-mer*** is a substring of length $\boldsymbol{k}$ of a read, so a read of length n contains $n-k+1$ k-mers. A ***minimizer*** of length $\boldsymbol{p}$ of a k-mer is the lexicographically smallest length-p substring of the k-mer. Finally, a ***super-mer*** is generated by merging consecutive k-mers in the read that have the same minimizer. Two k-mers are consecutive in a read if their starting positions in the read are consecutive. Figure 1 illustrates the super-mer generation algorithm (Algorithm 1) with a simple example.

Input: $k=9, p=5, n=14$			**Super-mer Generation (mimimizers are underlined)**
	Read =	CAAATTACTGCATA	
i=0	(k-mer #1)	CAAATTACT	*minimizer* ← AAATT, *minimizer_beg_pos* ← 1, *supermer_beg_pos* ← 0
i=1	(k-mer #2)	AAATTACTG	no update on minimizer
i=2	(k-mer #3)	AATTACTGC	*minimizer* ← *new_minimizer* ← AATTA, *minimizer_beg_pos* ← 2
i=2	(*super-mer* #1)	CAAATTACTG	generate current super-mer, *supermer_beg_pos* ← 2
	super-mer #1 is made up of k-mer #1 and #2, minimizer = AAATT		
i=3	(k-mer #4)	ATTACTGCA	*minimizer* ← *new_minimizer* ← ACTGC, *minimizer_beg_pos* ← 7
i=3	(*super-mer* #2)	AATTACTGC	generate current supermer, *supermer_beg_pos* ← 3
	super-mer #2 is made up of k-mer #3 only, minimizer = AATTA		
i=4	(k-mer #5)	TTACTGCAT	no update
i=5	(k-mer #6)	TACTGCATA	no update
	(*super-mer* #3)	ATTACTGCATA	*supermer_beg_pos* $\neq n-k$, generate the last super-mer
	super-mer #3 is made up of k-mer #4 #5 #6, minimizer = ACTGC		

Figure 1: Illustration of super-mer generation (The minimizers are underlined)

III. Sequential Algorithm

```
Algorithm 1: Super-mer Generation (sequential)
Input:  read string $\boldsymbol{S} = s_0, s_1, s_2, \dots, s_{n-1}$
        k-mer length $\boldsymbol{k}$, minimizer length $\boldsymbol{p}$
1.  Procedure GenerateSupermer ($\boldsymbol{S}, \boldsymbol{k}, \boldsymbol{p}$):
2.      $minimizer$ = the minimum $\boldsymbol{p}$-substring of $\boldsymbol{S}[\mathbf{0}, \boldsymbol{k} - \mathbf{1}]$
3.      $minimizer\_beg\_pos$ = the starting position of minimizer in $\boldsymbol{S}$
4.      $supermer\_beg\_pos = 0$
5.      for $i$ from $\mathbf{1}$ to $n - k$ do:
6.          if $i > minimizer\_beg\_pos$ then:
7.              $new\_minimizer$ = the minimum $\boldsymbol{p}$-substring of $S[i, i + k - 1]$
8.              update $minimizer\_beg\_pos$
9.          else:
10.             if $S[i + k - p, i + k - 1] \leq minimizer$ then:
11.                 $new\_minimizer = S[i + k - p, i + k - 1]$
12.                 update $minimizer\_beg\_pos$
13.             end if
14.         end if
15.         if $new\_minimizer \neq minimizer$ then:
16.             $minimizer = new\_minimizer$
17.             save current super-mer: $\boldsymbol{S}[supermer\_beg\_pos, i + k - 1]$
18.             $supermer\_beg\_pos = i$
19.         end if
20.     end for
21.     if $supermer\_beg\_pos \neq n - k$ then:
22.         save the last super-mer: $\boldsymbol{S}[supermer\_beg\_pos, n]$
23.     end if
24. end procedure
```

IV. Your Implementation Task

The code skeleton **"gensupermer_cuda.cu"** is provided, in which the function "**GenerateSupermer_GPU**" calls the two kernel functions that you will implement:

1. The first kernel function "**GPU_GenMinimizer**":
   __global__ void GPU_GenMinimizer(_in_ _out_ T_GPU_data data, int K_kmer, int P_minimizer)

In this function, $data.reads$ is given in the CSR format, and the function will output $data.minimizers$. You can choose to use a 2bits-compressed format to store each minimizer (See Notes at the end of Section IV) or use your own method.

2. The second kernel function "**GPU_GenSKM**":

```
__global__ void GPU_GenSKM(_in_ _out_ T_GPU_data data, int K_kmer, int P_minimizer)
```

In this function, $data.minimizers$ is given (generated by calling the kernel $\boldsymbol{GPU\_GenMinimizer}$) and the function will output $data.supermer\_offs$. The starting positions of all supermers in their corresponding reads are stored in $data.supermer\_offs$. The last element of $data.supermer\_offs$ will be n-k+1 to indicate the end of this array, where n is the length of the read and k is the length of kmer.

Notes:

1. **The CSR Format:** The CSR (compressed sparse row) format stores all rows in a single data array and has an offset array to point to the beginning position of each row in the data array. For example, we store all reads in the array $read\_CSR$, and the index of each read in the array $read\_CSR\_offset$. Figure 2 shows an example of four reads `AC`, `ACT`, `AG`, and `GCT` stored in CSR.

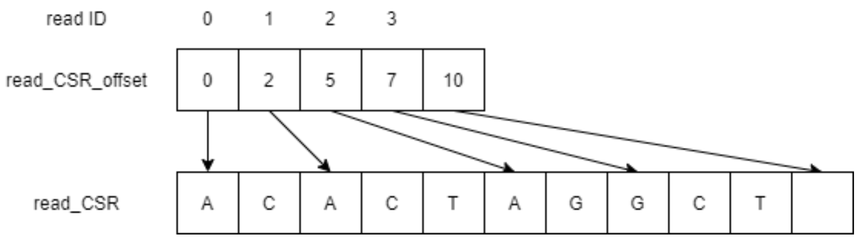

Figure 2: An Example of CSR Format

2. **2bits-compression of minimizers**: You can store a minimizer in a 32-bit unsigned integer for efficient comparison and access. In this format, each base ('A', 'C', 'T', 'G') in the minimizer is represented as a 2-bit integer (0,1,2,3 or 0b00, 0b01, 0b10, 0b11 accordingly). We provide the "d_basemap" constant array in the GPU memory for you to convert a minimizer string into an unsigned integer in the GPU kernel program **GPU_GenMinimizer**. For example, a minimizer (p=16) ACACACAGAGAGATTT can be represented as an unsigned int 0b00010001000100110001000100011111111, and a minimizer (p=5) AACGT as an unsigned int 0b0000011011.
3. The struct "T_GPU_data" contains the following members: the "reads" array is the data array in the CSR format of the reads; the "reads_offs" array is the offset array in the CSR format of the reads; the "minimizers" array will be output in the kernel "**GPU_GenMinimizer**" and used as input in the kernel "**GPU_GenSKM**"; the "supermer_offs" array is the output of the kernel "**GPU_GenSKM**" and stores the starting positions of all supermers in their corresponding reads. In the following example, the minimizer of the second read "CAAACCTTGG" is "1,1,5" because the minimizer of the first kmer "CAAACCTT" is "AAAC", which is expressed in 2bits-compression as "0b00000001" = 1. The minimizer of the third kmer "AACCTTGG" is "AACC": "0b00000101" = 5. The supermer_offs in read 2 is 0, 2, 3. The first two elements 0 and 2 are the starting positions of the two supermers, and the last element 3 is n -k+1, where n is the length of the read, 10, and k is 8.

   - Example (2 reads, k=8, p=4)
     - `*reads:         AAAACCTT CAAACCTTGG`
     - `*reads_offs:    0, 8, 18`
     - `*minimizers:    0, -, -, -, -, -, -, -, 1, 1, 5, -, -, -, -, -, -, -`
     - `*supermer_offs: 0, 1, -, -, -, -, -, -, 0, 2, 3, -, -, -, -, -, -, -`

     The last item in each offset sequence is $n - k + 1$ *(the underlined item).*

V. The Given Package

gensupermer _cuda.cu	The CUDA program skeleton for you to fill in the missing code of the two GPU kernel functions.
gensupermer.hpp	The header file for gensupermer_cuda.cu
main.cpp	The main function of the CUDA version program.
gensupermer _sequential.cpp	The sequential version of super-mer generation. It is for your reference and result comparison.
dataset/*.txt	Each text file is a test dataset with each line containing a read.
result/	An empty folder for super-mer output.
utilities.hpp	A header file containing utility functions such as file loading, result output, and result comparison.

VI. Compile and Run

Notes:

1. *We will **not** apply any compiler optimization during the assessment (e.g., -O2 -O3 ...).*
2. ***No** external library or header file is allowed (e.g., Boost or other third-party libraries).*

An example of input file and corresponding output file:

The input file contains multiple lines of reads. The following is an example containing 5 reads.

```
GATAACGAGTTCTAGAAAACTGGGTCCC
AGATCAGCAAACCCTGAGAAAAA
AGAAAAATTAGCAATAATTAGCAGTGTTCATAACA
AGAAGACAACTGGGCCCGGGGGAC
GAGGGAGAACCTGATTTCCAGAGT
```

Example input: dataset1.txt

The output will be the super-mers generated from the reads. The order of the super-mers in the output file is lexicographic. For example, the 7th and the last super-mers in the example output file are generated from the first read in the example input file. The 7th super-mer contains 7 consecutive k-mers. These 7 k-mers share the same minimizer AAAACTG.

```
AAAATTAGCAATAATTAGCAG
AAATTAGCAATAATTAGCAGT
AATTAGCAATAATTAGCAGTGTTCATAA
AGAAAAATTAGCAATAATTAGCA
AGAAGACAACTGGGCCCGGGGGAC
AGATCAGCAAACCCTGAGAAAAA
ATAACGAGTTCTAGAAAACTGGGTCCC
ATAATTAGCAGTGTTCATAACA
GAGGGAGAACCTGATTTCCAGAGT
GATAACGAGTTCTAGAAAACT
```

Example output (k=21, p=7): my_gs_output.txt

Example of compiling and running the sequential program:

```
# Compile:
g++ gensupermer_sequential.cpp -o seq_gs -std=c++11
# Run and save super-mers to text file:
# (e.g., k=21, p=7, data file is ./dataset/dataset1.txt, and result file is ./result/seq_gs_output.txt):
./seq_gs 21 7 ./dataset/dataset1.txt ./result/
# Run without saving super-mers to text file:
./seq_gs 21 7 ./dataset/dataset1.txt
```

Example of compiling and running your CUDA program:

```
# Compile:
nvcc -std=c++11 gensupermer_cuda.cu main.cpp -o cuda
# Run:
./cuda 21 7 8 512 ./dataset/dataset1.txt <correctness_check> <result_folder
(optional)>
# <correctness_check> can be 0 or 1. If 1, the program will automatically check the
correctness by comparing all_supermers with the sequential version. <result_folder> is
optional; if not provided, the super-mers will not be saved to any file.
```

VII. Submission and Evaluation

You only need to submit your completed ***gensupemer_cuda.cu*** to Canvas before the deadline. You can add your code only in the specified section. Your program should not have any extra output to the result file or the standard output.

We will evaluate your program in different parameter settings. We will vary p and k ($10 \leq 2p \leq k < 23$). The length of each read is greater than 22. Both the total length of all reads and that of the generated super-mers will be less than INT_MAX (0x7FFFFFFF). We will test with different numbers of threads (1 ≤ num_blocks_per_grid ≤ 32 and 32 ≤ num_threads_per_block ≤ 512). We will prepare multiple sets of configurations and datasets. Your grade will be marked based on both correctness and running time.