COMP5112 / MSBD5009 Parallel Programming (Spring 2022)

Assignment 1: MPI Programming

Submission deadline: **23:59 (pm) on Mar. 16 (Wednesday), 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 MPI 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 k of a read, so a read of length n contains $n-k+1$ k-mers. A ***minimizer*** of length 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	$minimizer \leftarrow$ AAATT, $minimizer\_beg\_pos \leftarrow 1$, $supermer\_beg\_pos \leftarrow 0$
i=1	(k-mer #2)	AAATTACTG	no update on minimizer
i=2	(k-mer #3)	AATTACTGC	$minimizer \leftarrow new\_minimizer \leftarrow$ AATTA, $minimizer\_beg\_pos \leftarrow 2$
i=2	(*super-mer* #1)	CAAATTACTG	generate current super-mer, $supermer\_beg\_pos \leftarrow 2$
	super-mer #1 is made up of k-mer #1 and #2, minimizer = AAATT		
i=3	(k-mer #4)	ATTACTGCA	$minimizer \leftarrow new\_minimizer \leftarrow$ ACTGC, $minimizer\_beg\_pos \leftarrow 7$
i=3	(*super-mer* #2)	AATTACTGC	generate current supermer, $supermer\_beg\_pos \leftarrow 3$
	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 $\boldsymbol{S} = s_0, s_1, s_2, \ldots, s_{n-1}$
k-mer length $\boldsymbol{k}$, minimizer length $\boldsymbol{p}$

Procedure GenerateSupermer ($\boldsymbol{S}, \boldsymbol{k}, \boldsymbol{p}$):
  $minimizer$ = the minimum $\boldsymbol{p}$-substring of $\boldsymbol{S}[\boldsymbol{0}, \boldsymbol{k}-\boldsymbol{1}]$
  $minimizer\_beg\_pos$ = the starting position of minimizer in $\boldsymbol{S}$
  $supermer\_beg\_pos = 0$
  for i from $\boldsymbol{1}$ to $n-k$ **do:**
    if $i > minimizer\_beg\_pos$ **then**:
      $new\_minimizer$ = the minimum $\boldsymbol{p}$-substring of $S[i, i+k-1]$
      update $minimizer\_beg\_pos$
    else:
      if $S[i+k-p, i+k-1] \leq minimizer$ **then**:
        $new\_minimizer = S[i+k-p, i+k-1]$
        update $minimizer\_beg\_pos$
      end if
    end if
    if $new\_minimizer \neq minimizer$ **then:**
      $minimizer = new\_minimizer$
      save current super-mer: $\boldsymbol{S}[supermer\_beg\_pos, i+k-1]$
      $supermer\_beg\_pos = i$
    end if
  end for
  if $supermer\_beg\_pos \neq n-k$ **then:**
    save the last super-mer: $\boldsymbol{S}[supermer\_beg\_pos, n]$
  end if
end procedure

IV. Your Implementation Task

The code skeleton **"gensupermer_mpi.cpp"** is provided. In this assignment, your task is to complete the missing code in the specified section in the main function:

1. Scatter the read data to each MPI processes.
2. Perform the super-mer generation in each process (you can call the provided ***read2supermers***).
3. Gather all the super-mers to Process 0, and store them in a vector of strings (variable "***all_supermers***" in which each super-mer is a string). The order of the super-mers in the vector is unimportant.

Notes:

1. **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.

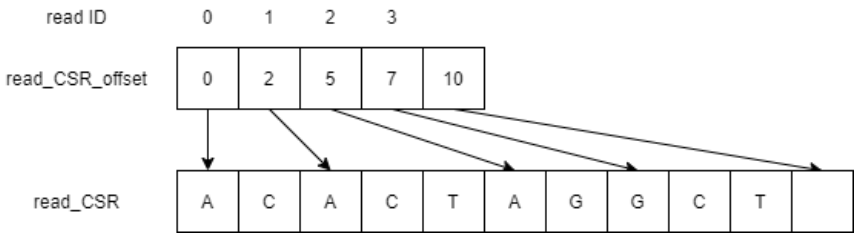

Figure 2: An Example of CSR Format

2. We provide the function: ***read2supermers***. The function takes one read as input, generates all super-mers in the read, and stores them in a vector<string>. You can reference its usage in the sequential code.

3. For more information on MPI APIs, you may refer to https://www.open-mpi.org/doc/v4.0/.

V. The Given Package

gensupermer_mpi.cpp	The MPI program skeleton for you to fill in the missing code in the designated place.
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 MPI program:

```
# Compile:
mpic++ -std=c++11 gensupermer_mpi.cpp -o gs
# Run:
mpiexec -n <num_process> ./gs <k> <p> <data_file> <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_mpi.cpp*** 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). The number of MPI processes will be set to values less than 16. We will prepare multiple sets of evaluation arguments and datasets. Your grade will be marked based on both correctness and best speedup achieved.