COMP5112 / MSBD5009 Parallel Programming (Spring 2025)

Assignment 1: MPI Programming

Submission deadline: **23:59 (pm) on 17 Mar 2025**

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. AI-generated code is not allowed. 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. All submission will be compiled and tested uniformly on given machines of the course (CSE Lab 2 machines).
5. Please direct any inquiries about this assignment to the designated TAs listed on the CANVAS assignment page.
6. Submit your assignment through Canvas before the deadline. **No late submission will be accepted.**

II. Problem Description

This assignment is on the MPI parallelization of a string processing task. The input of the task is multiple DNA fragment sequences (called ***reads***) of variable lengths. The characters in the reads only include 'A', 'C', 'G', and 'T' (4 types of nucleotides). A ***k-mer*** is a substring of length k of a read, so a read of length l contains $l-k+1$ k-mers. ***Universal minimizers*** with a given hash function and an encoding function are the k-mers whose encoded values are hashed to 0. The task is to generate the list of all ***universal minimizers*** for each ***read***. The universal minimizers in the list are in the same partial order as in the read. Figure 1 shows an example read, its k-mers, and the expected result -- a list of encoded universal minimizers of the input read:

Given:

$k = 8$, hash function: $h(f(S)) = f(S)\ \%\ 7$. *f(S) is the encoding function to convert a k-mer into integer.*

Read: `CAAATTACTGCATAG`

k-mer **(*S*)**		***Encoded*** **($f(S)$)**	***Hash value*** **($h(f(S))$)**	***Is universal minimizer?***
(k-mer #1)	`CAAATTAC`	16625	**0**	**Y**
(k-mer #2)	`AAATTACT`	967	1	N
(k-mer #3)	`AATTACTG`	3870	6	N
(k-mer #4)	`ATTACTGC`	15481	4	N
(k-mer #5)	`TTACTGCA`	61924	2	N
(k-mer #6)	`TACTGCAT`	51091	5	N
(k-mer #7)	`ACTGCATA`	7756	**0**	**Y**
(k-mer #8)	`CTGCATAG`	31026	2	N

Result (list of integer-encoded universal minimizers of the input read, in the same partial order as in the read)**:**

16625 7756

Figure 1: Example of generating the list of universal minimizers for a given read

Integer encoding of a k-mer:

Let $S = s_1 s_2 s_3 \ldots s_k$ be a k-mer, where each $s_i \in \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$ represents a nucleotide at position i.

The function $f(S)$ maps the k-mer S to an integer by mapping each nucleotide to a number: A=0, C=1, G=2, T=3. Then we represent the encoding in the binary form:

string:	A	C	C	G	T
$v(s_i)$:	0	1	1	2	3
binary:	00	01	01	10	11

We concatenate the binary numbers and obtain the output: $0001011011_{(2)} = 91_{(10)}$.

Figure 2: Integer encoding of a k-mer

III. Sequential Algorithm

Algorithm 1: Converting read to universal minimizer list (sequential)

Input: **A** read string $\boldsymbol{R}$ of length $\boldsymbol{l}$, characters are indexed from 0 to $l-1$.
k-mer length $\boldsymbol{k}$.

Given functions: hash function $h(x)$; integer encoding function $f(S)$.

```
1.  Procedure UniversalMinimizerListGeneration (R, l, k):
2.      res = []        // result list for universal minimizers
3.      for i from 0 to l − k do:
4.          S = R[i: i + k − 1]        // S: current k-mer, substring of R
5.          S_encoded = f(S)           // integer, encoded k-mer
6.          H = h(S_encoded)           // hash value of current k-mer
7.          if H == 0 then:
8.              res.append (S_encoded)
9.          end if
10.     end for
11.     return res
12. end procedure
```

IV. Input and Output

We provide test input files named "input_*.txt", *=1,2,3.., and their corresponding output files "s_output_*.txt". You can run the script "result_check.sh" in the assignment package to compare your output with the expected output to check the correctness.

Input:

The input of the program is a text file of parameters and reads. The first line contains a single integer, k, which is the k-value (length) of k-mer, $\boldsymbol{4 \le k \le 16}$. In the next n lines till the end of file ($1 \le n \le 100000$), each line is a read (a string that contains only 'A', 'C', 'G', and 'T') with a variable length l, $k \le l \le 10000$.

Output:

The output contains n lines, n is the number of input reads. Each line contains one or multiple integers, separated by a space. If the corresponding input read has no universal minimizer, the output line contains a single integer "-1"; otherwise, the output line is a list of universal minimizers in the integer-encoded format of the corresponding input read.

* The order of the lines in the output file is the same as the reads in the input file. For example, the first line of the output file is the universal minimizer list of the first read (the read in the second line of the input file); the second line of the output file is the universal minimizer list of the second read (the read in the third line of the input file), and so on.
* The order of the universal minimizers in a line (list) in the output file is the same partial order as that in the read (i.e., any two universal minimizers in an output line are in the same partial order as them in the input read).

Example input file and corresponding output file:
The following is an example input file giving the length of k-mer 8 and containing 4 reads.

```
8
CAAATTACTGCATAG
AAATTACTGCAT
AAATTACTGCATA
AAAAAAAAAA
```

Example input file: input_1.txt

The following is the corresponding output with the example input.

```
16625 7756
-1
7756
0 0 0
```

Example output: s_output_1.txt

In the output file, the number of lines is equal to the number of input reads. The first line of the output is the universal minimizer list of the first read. As illustrated in Figure 1, the first k-mer (k=8) `CAAATTAC` and the second to the last k-mer `ACTGCATA` of the first read are universal minimizers, and the two universal minimizers are encoded as integers and output in the first line of the output file. The second input read has no universal minimizer, so the second line in the output file is -1. The third read has a single universal minimizer. In the last read, all three reads (a read of length 10 has $l - k + 1 = 10 - 8 + 1 = 3$ k-mers) are universal minimizers so the last line of output contains three space-separated zeros (the encoded value (f("AAAAAAAA") $=$ 0).

V. Your Implementation Task

In this assignment, your task is to complete the function **`gen_um_lists_MPI`** in the provided code skeleton file **"your_mpi_implementation.hpp"**. You can only add your code in this file and are not allowed to modify any other files provided.

In your **`gen_um_lists_MPI`** function implementation, you need to do the following steps:

1. Scatter the read data to multiple MPI processes.
2. Perform universal minimizer list generation in each process (you can call the provided function: **`generate_universal_minimizer_list`**).
3. Gather all generated universal minimizer lists to Process 0.
4. Organize and order the lists properly, and store them into `std::vector<std::vector<kmer_t>> um_lists` in process 0.

*** Make sure the order of universal minimizer lists is the same as that of input reads.** E.g., for the i-th input read (i starts from 0 to n-1), its universal minimizer list should be also saved at the same index i in the result vector (`um_lists[i]`).

Notes and References:

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 3 shows an example of four reads `AC`, `ACT`, `AG`, and `GCT` stored in CSR. In the given code skeleton, we load all reads from the input file to a vector of strings and copy and store all the reads into the CSR format. You can either use the CSR or the vector of string as you wish when scattering reads. You may also find the CSR format useful when sending universal minimizers back to Process 0. Some useful functions (e.g., **`Vector2CSR`** and **`CSR2Vector`**) are provided in "utilities.hpp". You may choose to use them in your implementation.

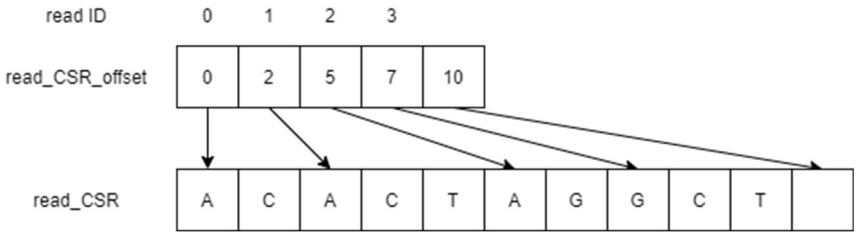

Figure 3: An Example of CSR Format

2. We provide the function: "**`generate_universal_minimizer_list`**", which returns a universal minimizer list of the given input read. You can use it in your implementation or write your own version.
3. We provide the function: "**`kmer_encoding`**." The function takes a k-mer string as input and returns an unsigned integer which is the k-mer's integer-encoded value.
4. We provide the hash function "**`int hash(kmer_t)`**".
5. For more information on MPI APIs, you may refer to https://www.open-mpi.org/doc/v5.0/.

VI. The Given Package

universal_minimizer_mpi.cpp	The MPI program skeleton includes the main function.
your_mpi_implementation.hpp	The **only** file you can write your code in and submit. You should implement the function "`gen_um_lists_MPI`" in this file and keep the function signature unchanged. This function will be called by the main function. You can add your own functions in this file and include other C++ standard libraries in this file.
utilities.hpp	A header file containing utility functions such as file loading and result output. The universal minimizer list generation algorithms are also in it. You can use these provided functions.
universal_minimizer_sequential.cpp	The sequential version of universal minimizer list generation. It is for your reference and result comparison.
datasets/input_*.txt	Each text file is a test dataset. For details, please refer to section IV.
datasets/s_output_*.txt	Each text file "s_output_i.txt" is the expected output of "input_i.txt".

output.txt	The output of the sequential program and that of your MPI program. You **should not** change the given output filename in the program skeleton.
result_check.sh	A script that accepts a parameter x, to compare if output.txt and file x are identical.

VII. Compile and Run

Notes:

1. *We will apply -O2 compiler optimization during the assessment.*
2. ***No*** *external library or header file is allowed (e.g., lib-boost or other third-party libraries).*

Example of compiling and running the sequential program with an input file:

```
# Compile:
g++ universal_minimizer_sequential.cpp -o seq_um -O2 -std=c++11
# Run (the output file output.txt will be generated in the current directory):
./seq_um ./datasets/input_1.txt
```

Example of compiling and running your MPI program with an input file:

```
# Compile:
mpic++ -std=c++11 universal_minimizer_mpi.cpp -o mpi_um -O2 -std=c++11
# Run (the output file output.txt will be generated in the current directory):
mpiexec -n <num_process> ./mpi_um ./datasets/input_1.txt
```

Example of checking if your output is correct compared with s_output_1.txt:

```
# Run in bash:
./result_check.sh ./datasets/s_output_1.txt
# It will report if your output.txt is correct (the same as s_output_1.txt)
```

VIII. Submission and Evaluation

You only need to submit your completed **"your_mpi_implementation.hpp"** to Canvas before the deadline. You can only write your code in this file. Your program should not have any extra output to any other file.

We will evaluate your program on multiple datasets. We may change the hash function modular number in our evaluation so please make sure you don't write and use your own hash function in your implementation. The range of k (length of k-mer), n (number of reads), and l_i (length of read) are described in Section IV. The numbers of MPI processes will be set to values no greater than 32. We will prepare multiple sets of evaluation parameters and datasets. Your submitted code will be marked based on both correctness and speedups.