# Parallel Computing – Final Project
# Batch 2022 – Year 2025

## String-Matching Algorithm Parallelization

Short Description: Choose any String-Matching Algorithm, make it parallel by using CUDA Programming, then make a comparison analysis of parallel process between 2 algorithms. There are 5 options of Algorithms you can take (**take only 2 of them**):

- Knuth-Morris Pratt Algorithm
- Needleman-Wunsch Algorithm
- Smith-Watermann Algorithm
- Aho-Corasick Algorithm
- Rabin Karp Algorithm

Dataset: Sample DNA Sequence (Nucleotide).

## Paper Report (Due: Tuesday, 28 April 2025)

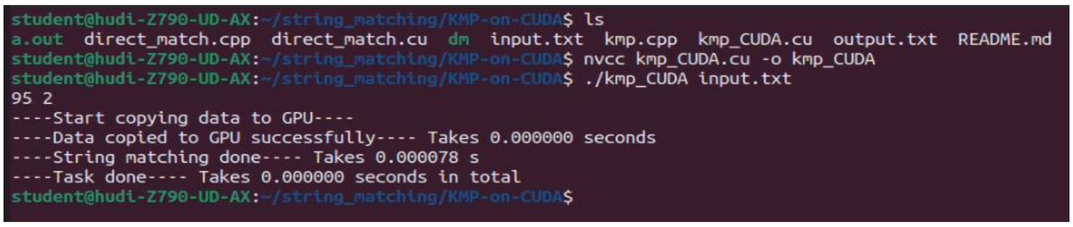
IEEE Paper Format Report. (Find the format here: IEEE Paper Template)

## Research Guide

1. [C/C++] Find the code and dataset for each algorithm on following links:
   a. Knuth-Morris Pratt (KMP)
   b. Needleman-Wunsch (NW)
   c. Smith-Watermann (SW)
   d. Aho-Corasick (AC)
   e. Rabin Karp (RK)

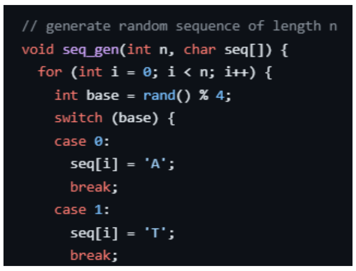
   Dataset (Optional): https://www.ncbi.nlm.nih.gov/datasets/genome/GCF_000001635.27/

2. Instruction to compile and run the code file could be found in these directories:
   (Sample to run KMP-CUDA, please take notes that this source code is downloaded beforehand, if you do not find your algorithm in the folder, please download it first from the given github link)

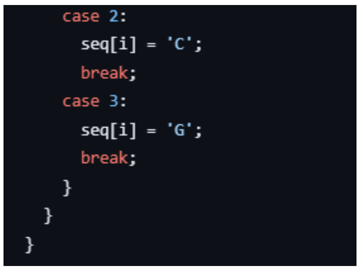


```
student@hudi-Z790-UD-AX:~/string_matching/KMP-on-CUDA$ ls
a.out  direct_match.cpp  direct_match.cu  dm  input.txt  kmp.cpp  kmp_CUDA.cu  output.txt  README.md
student@hudi-Z790-UD-AX:~/string_matching/KMP-on-CUDA$ nvcc kmp_CUDA.cu -o kmp_CUDA
student@hudi-Z790-UD-AX:~/string_matching/KMP-on-CUDA$ ./kmp_CUDA input.txt
95 2
----Start copying data to GPU----
----Data copied to GPU successfully---- Takes 0.000000 seconds
----String matching done---- Takes 0.000078 s
----Task done---- Takes 0.000000 seconds in total
student@hudi-Z790-UD-AX:~/string_matching/KMP-on-CUDA$
```

3. Modify the dataset to fit the input format for each code file. Dataset per-processing is important to differentiate the pattern and data banks as these datasets will be measured by its size. **List of the sequences** is recommended to be **generated manually** from a function thus the length and variation can be controlled. Refer to this generator function in case you need it:



```
// generate random sequence of length n
void seq_gen(int n, char seq[]) {
  for (int i = 0; i < n; i++) {
    int base = rand() % 4;
    switch (base) {
    case 0:
      seq[i] = 'A';
      break;
    case 1:
      seq[i] = 'T';
      break;
```

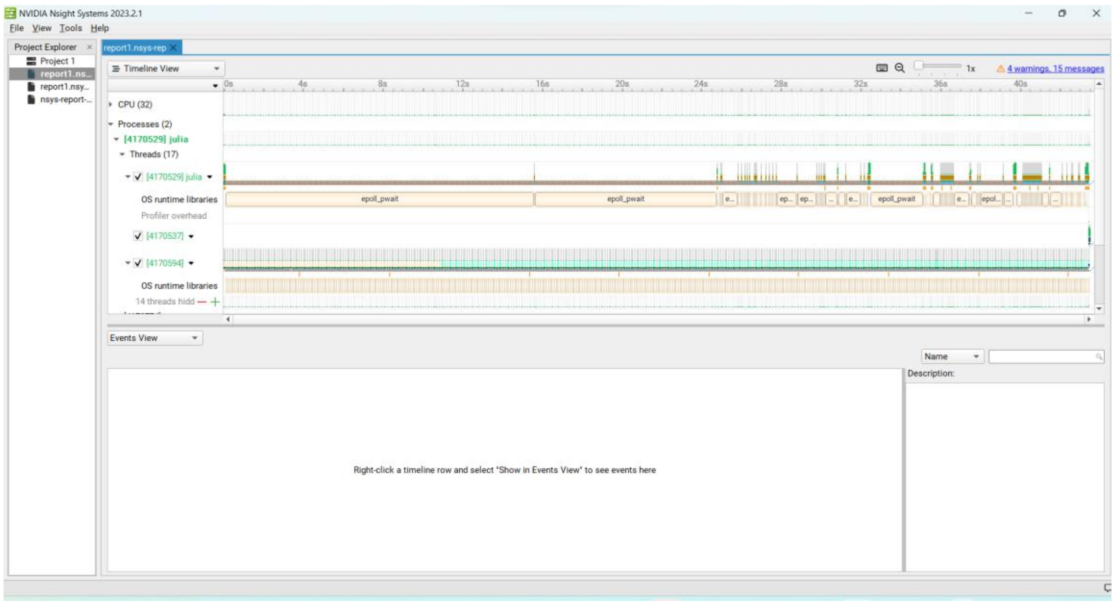


```
    case 2:
      seq[i] = 'C';
      break;
    case 3:
      seq[i] = 'G';
      break;
    }
  }
}
```

The starting point of the amount of pattern is suggested to be 16, representing the starting point of $2^n$ sequences where n=4. Furthermore, please record the performance of each algorithm as n is increased. The length of patterns is suggested to be 10 characters long.
References for code file is taken from Github, thus, it is not guaranteed that you can execute the whole project right away off the bat. Please do make some adjustments to the code so that the server could initiate run (whether from makefile or manual execution with flags and arguments, consider readme of each repository as your main guide to run the project.)

4. Make sure the execution is tracked and labelled with the correct sequence size so that you will not lose track of the profiling later. Check on the algorithm correctness as well.
   Sample Execution Command:
   ```
   nvcc kmp_8192.cu -o kmp_8192
   ./kmp_8192 10 >> result_kmp_8192_10.txt
   ```
5. Conduct profiling with this command (make sure you are in the same directory as your code file):
   [C/C++]
   ```
   nsys profile  -o report --stats=true ./<program_name>.exe
   ```
   Which generates files: report.nsys-rep, change the filename if needed.

6. Open the *nsys-rep* files using NVIDA Nsight Systems (use the version according to your CUDA Toolkit), it should look like this:



Right-click a timeline row and select "Show in Events View" to see events here

7. Take the data of execution time and memory consumption data of each kernel, and for the whole process of the application, put it into the table then convert the table into graph for a better analysis.
   <u>Table sample</u>:

Dataset Length: $2x10^{10}$; Pattern Length: 10

| n | Number of Pattern ($2^n$) | Sequential (kmp.cpp) | CUDA (kmp_CUDA.cu) | | Speedup |
|---|---|---|---|---|---|
| | | Running Time | Running Time | Memory Consumption | |
| 3 | 8 | | | | |
| 4 | 16 | | | | |
| 5 | 32 | | | | |
| 6 | 64 | | | | |
| 7 | 128 | | | | |
| 8 | 256 | | | | |
| 9 | 512 | | | | |
| 10 | 1024 | | | | |

8. Following are guidelines for each chapter title for the final report:
   - *Abstract*
   - Chapter I – Introduction
   - Chapter II – CUDA and String Matching
   - Chapter III – Methodology
   - Chapter IV – Result and Analysis
   - Chapter IV – Conclusion