PARALLEL IMPLEMENTATION OF BOYER-MOORE-HORSPOOL ALGORITHM FOR STRING MATCHING USING OPENMP, MPI AND CUDA.

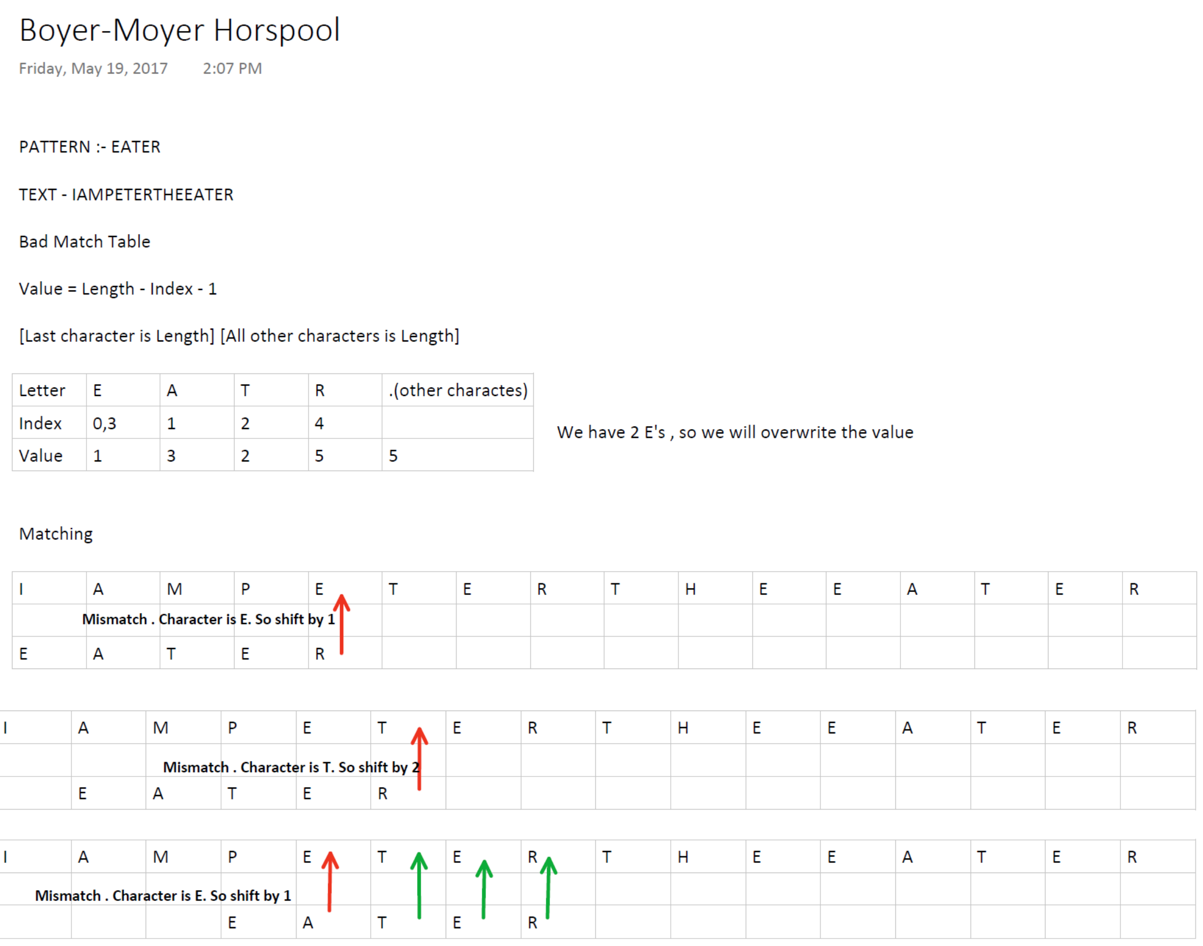
Ramkumar Rajabaskaran: 85241493
Sindhuri Rayavaram: 28562782

Problem Statement:

The Boyer-Moore-Horspool algorithm is an algorithm for finding substrings in strings.

Sequential Algorithm:

The Boyer-Moore algorithm suggested to move the index of the string by m places if there is a mismatch occurring at i-m place. In this algorithm, we can do the shifting based on the index of the last letter compared. We do the comparison from right to left of the pattern. We maintain a table called Bad-Match table to see how many values to shift when a mismatch occurs. This algorithm has an average running time of $O(m)$ where
M is the size of the pattern. We have explained the algorithm below.



Boyer-Moyer Horspool

Friday, May 19, 2017 2:07 PM

PATTERN :- EATER

TEXT - IAMPETERTHEEATER

Bad Match Table

Value = Length - Index - 1

[Last character is Length] [All other characters is Length]

Letter	E	A	T	R	.(other charactes)
Index	0,3	1	2	4	
Value	1	3	2	5	5

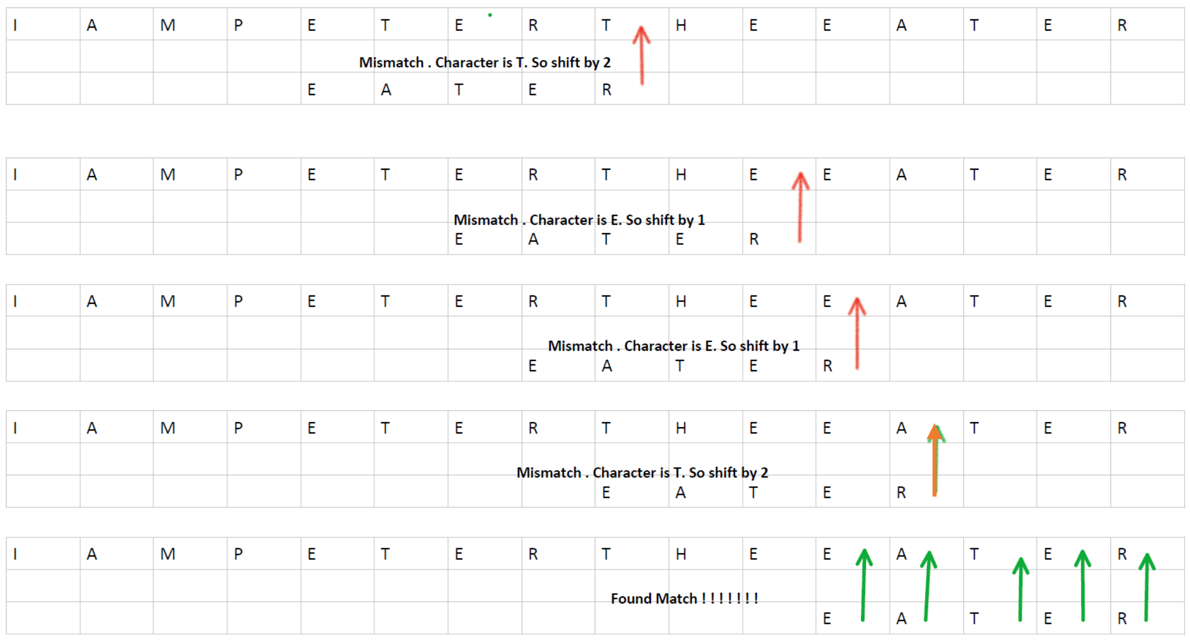
We have 2 E's , so we will overwrite the value

Matching

I	A	M	P	E	T	E	R	T	H	E	E	A	T	E	R
	Mismatch . Character is E. So shift by 1														
E	A	T	E	R											

I	A	M	P	E	T	E	R	T	H	E	E	A	T	E	R
		Mismatch . Character is T. So shift by 2													
	E	A	T	E	R										

I	A	M	P	E	T	E	R	T	H	E	E	A	T	E	R
Mismatch . Character is E. So shift by 1															
			E	A	T	E	R								


I A M P E T E R T H E E A T E R
Mismatch . Character is T. So shift by 2
E A T E R
Mismatch . Character is E. So shift by 1
Mismatch . Character is E. So shift by 1
Mismatch . Character is T. So shift by 2
Found Match ! ! ! ! ! ! !

We have implemented in C++. We used two functions. First function implements the Bad-Match table. This is the preprocessing phase where we use extra memory to hold information about the pattern to reduce the running time. The Second function does the String Matching based on our algorithm.

PARALLEL ALGORITHM:
We can parallelize this code by dividing the input text into chunks and giving each chunk to a particular thread. We divide the work carefully so as to ensure that there is no error in matching and each portion is independent.

OPENMP IMPLEMENTATION:
We have divided the String into blocks between different threads. We index a thread so that we each thread takes distinct parts. We have set an offset to avoid the cases where the pattern is present in two different blocks.

MPI IMPLEMENTATION:
In this we are dividing the work between different processes and the communication is happening using scatter and gather.

CUDA IMPLEMENTATION:
In this implementation, we are dividing the work based on the number of blocks and threads per block and each thread will start from a unique position in the string based on the thread id.

In all these implementations, we are taking care as to avoid any error based on overlapping or improper splitting of the text.

PROGRESS:
We have finished the Sequential code, OpenMP implementation. We are working on MPI, CUDA codes and to provide analytics.