%%cu

#include<iostream>

#include <string>

#include <string>

#include <vector>

#include <list>

#include <iostream>

#include <chrono>

#include <cstdlib>

const char ETX = '$';

\_\_global\_\_ void generate\_table(int\* table, int table\_size, int n) {

int index = blockIdx.x \* blockDim.x + threadIdx.x;

int stride = blockDim.x \* gridDim.x;

for(int i = index; i <table\_size; i+=stride) {

if( i<n) {

table[i] = i;

} else {

table[i] = -1;

}

}

}

\_\_device\_\_ bool compare\_rotations(const int& a, const int& b, char\* genome, int n) {

if (a< 0) {

return false;

}

if (b < 0) {

return true;

}

for(size\_t i = 0; i < n; i++) {

if (genome[(a + i) % n] != genome[(b + i) % n]) {

return genome[(a + i) % n] < genome[(b + i) % n];

}

}

return false;

}

\_\_global\_\_ void bitonic\_sort\_step(int\* table, int table\_size, int j, int k, char\* genome, int n) {

unsigned int i = threadIdx.x + blockDim.x \* blockIdx.x;

unsigned int ixj = i ^ j;

if(i < table\_size) {

if(ixj > i) {

if ((i && k) == 0) {

if (compare\_rotations(table[ixj], table[i], genome, n)) {

int temp = table[i];

table[i] = table[ixj];

table[ixj] = temp;

}

}

if ((i && k) != 0) {

if (compare\_rotations(table[i], table[ixj], genome, n)) {

int temp = table[i];

table[i] = table[ixj];

table[ixj] = temp;

}

}

}

}

}

\_\_global\_\_ void reconstruct\_sequence(int\* table, char\* sequence, char\* transformed\_sequence,

size\_t n) {

int index = blockIdx.x \* blockDim.x + threadIdx.x;

int stride = blockDim.x \* gridDim.x;

for(int i = index; i < n; i+=stride) {

transformed\_sequence[i] = sequence[(n + table[i] - 1) % n];

}

}

struct rotation {

int index;

char\* suffix;

};

int cmpfunc(const void\* x, const void\* y)

{

struct rotation\* rx = (struct rotation\*)x;

struct rotation\* ry = (struct rotation\*)y;

return strcmp(rx->suffix, ry->suffix);

}

int\* computeSuffixArray(char\* input\_text, int len\_text)

{

struct rotation suff[len\_text];

for (int i = 0; i < len\_text; i++) {

suff[i].index = i;

suff[i].suffix = (input\_text + i);

}

qsort(suff, len\_text, sizeof(struct rotation),

cmpfunc);

int\* suffix\_arr

= (int\*)malloc(len\_text \* sizeof(int));

for (int i = 0; i < len\_text; i++)

suffix\_arr[i] = suff[i].index;

return suffix\_arr;

}

char\* findLastChar(char\* input\_text,

int\* suffix\_arr, int n)

{

char\* bwt\_arr = (char\*)malloc(n \* sizeof(char));

int i;

for (i = 0; i < n; i++) {

int j = suffix\_arr[i] - 1;

if (j < 0)

j = j + n;

bwt\_arr[i] = input\_text[j];

}

bwt\_arr[i] = '\0';

return bwt\_arr;

}

int main()

{

char input\_text[] = "ABCD";

int len\_text = strlen(input\_text);

int\* suffix\_arr

= computeSuffixArray(input\_text, len\_text);

char\* bwt\_arr

= findLastChar(input\_text, suffix\_arr, len\_text);

std::cout<<"Input text:"<<input\_text<<std::endl;

std::cout<<"Burrows - Wheeler Transform:"<<bwt\_arr<<std::endl;

int \*B = (int\*)(bwt\_arr);

cudaEvent\_t start, stop;

float elapsedTime;

cudaEventCreate(&start);

cudaEventCreate(&stop);

cudaEventRecord(start, 0);

cudaEventRecord(stop, 0);

cudaEventSynchronize(stop);

cudaEventElapsedTime(&elapsedTime, start, stop);

return 0;

}