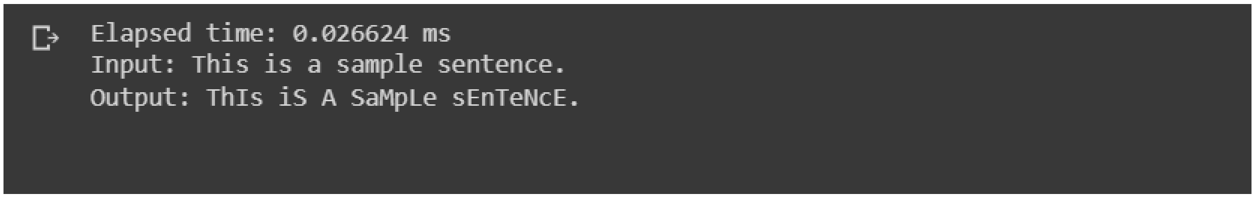
PRATAP SHINGANE
2020BTEIT00050
COURSE: PC

Converting the given sentence into 'Alternate (upper/lower) case text' (e.g. type -> TyPe) CUDA code

```
text' (e.g. type -> TyPe)

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

// Define device function to convert a character to
upper case
__device__ char cudaToUpperChar(char const& input)
{
    // Check if character is lowercase
    if (input >= 'a' && input <= 'z') {
        // Convert to uppercase and return
        return input - ('a' - 'A');
    } else {
        // Otherwise, return the character as is
        return input;
    }
}

// Define device function to convert a character to
lower case
__device__ char cudaToLowerChar(char const& input)
{
    // Check if character is uppercase
    if (input >= 'A' && input <= 'Z') {
        // Convert to lowercase and return
        return input + ('a' - 'A');
    } else {
        // Otherwise, return the character as is
        return input;
    }
}

// Define kernel function to alternate case of each
character in input string
__global__ void alternateCase(char *input, char
*output, int length)
{
    // Calculate index of current thread based on
block size and thread index
    int idx = blockIdx.x * blockDim.x + threadIdx.x;

    // Check if current index is within the length of
the input string
    if (idx < length)
    {
        // Determine if current character should be
converted to upper or lower case
        output[idx] = (idx % 2 == 0) ?
cudaToUpperChar(input[idx]) :
cudaToLowerChar(input[idx]);
    }
}

// Define main function
int main()
{
    // Define input sentence and get its length
    char sentence[] = "This is a sample sentence.";
    int length = strlen(sentence);

    // Allocate memory on the device for the input
and output strings
    char *d_input, *d_output, *h_output;
    cudaMalloc(&d_input, length * sizeof(char));
    cudaMalloc(&d_output, length * sizeof(char));

    // Copy input string to device memory
    cudaMemcpy(d_input, sentence, length *
sizeof(char), cudaMemcpyHostToDevice);

    // Set block and grid sizes for the kernel
    int block_size = 256;
    int grid_size = (length + block_size - 1) /
block_size;

     // To get the elapsed time
    cudaEvent_t start, end;
    cudaEventCreate(&start);
    cudaEventCreate(&end);

    cudaEventRecord(start);

    // Call the kernel with the specified block and
grid sizes
    alternateCase<<<grid_size, block_size>>>(d_input,
d_output, length);

     cudaEventRecord(end);
    cudaEventSynchronize(end);

    float elapsed_time_ms;
    cudaEventElapsedTime(&elapsed_time_ms, start,
end);

    printf("Elapsed time: %f ms\n", elapsed_time_ms);

    cudaEventDestroy(start);
    cudaEventDestroy(end);


    // Allocate memory on the host for the output
string and copy it from device memory
    h_output = (char *)malloc(length * sizeof(char));
    cudaMemcpy(h_output, d_output, length *
sizeof(char), cudaMemcpyDeviceToHost);

    // Print the input and output strings
    printf("Input: %s\n", sentence);
    printf("Output: %s\n", h_output);

    // Free memory on host and device
    free(h_output);
    cudaFree(d_input);
    cudaFree(d_output);

    return 0;
}
```

OUTPUT:



```
Elapsed time: 0.026624 ms
Input: This is a sample sentence.
Output: ThIs iS A SaMpLe sEnTeNcE.
```