```
!nvcc --version
!pip install git+git://github.com/andreinechaev/nvcc4jupyter.git
%load_ext nvcc_plugin
```

```
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2019 NVIDIA Corporation
Built on Sun_Jul_28_19:07:16_PDT_2019
Cuda compilation tools, release 10.1, V10.1.243
Collecting git+git://github.com/andreinechaev/nvcc4jupyter.git
  Cloning git://github.com/andreinechaev/nvcc4jupyter.git to /tmp/pip-req-build-keyi
  Running command git clone -q git://github.com/andreinechaev/nvcc4jupyter.git /tmp/
Building wheels for collected packages: NVCCPlugin
  Building wheel for NVCCPlugin (setup.py) ... done
  Created wheel for NVCCPlugin: filename=NVCCPlugin-0.0.2-cp36-none-any.whl size=430
  Stored in directory: /tmp/pip-ephem-wheel-cache-fyn5heze/wheels/10/c2/05/ca241da37
Successfully built NVCCPlugin
Installing collected packages: NVCCPlugin
Successfully installed NVCCPlugin-0.0.2
created output directory at /content/src
Out bin /content/result.out
```

```
%%cu
#include <iostream>
#include <cstdio>
#include <string>
#define TOGG(k) ( ( ( (k) & 1 ) << 5 ) )
#define RAN(charac) ( 65 + ( charac % 26 ) + TOGG ( charac ) )
using namespace std;

__global__
void RunLengthEncodingComputation (char *orig, int *_encoXst, int n) {
    int index = ( (blockIdx.x * blockDim.x) + threadIdx.x );
    index <<= 7;

    if(orig[index] == orig[index-1])
        while(index < n && orig[index] == orig[index-1])
            ++index;

    for (int i = index; i < fminf(index + 128, n); )
    {
        char temp = orig[i];
        int t_ = i;
        while (i < n && temp == orig[i])
            ++i;
        _encoXst[t_] = i;
    }
}

int main()  {
    int n=1000;
    char s[1000];
    fscanf(fopen("input.txt", "r"), "%s", s);
    char *cudas;
    int *_encoXst, *_inter = new int[n];

    int threads = (1 << 7);
    int blocks = ( ( n>>14 ) + ( ( n & ( (1<<14)-1 ) ) != 0 )  );

    cout << threads << " : " << blocks << endl;

    cudaMalloc (&cudas, n*sizeof(char));
    cudaMalloc (&cudas, n*sizeof(char));
    cudaMalloc (&_encoXst, n*sizeof(int));

    cudaMemcpy (cudas, s, n*sizeof(char), cudaMemcpyHostToDevice);

    RunLengthEncodingComputation <<<blocks, threads>>> (cudas, _encoXst, n);
    cudaDeviceSynchronize();
    cudaMemcpy(_inter, _encoXst, n*sizeof(int), cudaMemcpyDeviceToHost);

    string ans;
    int sum = 0;
    for(int i = 0; i < n; i = _inter[i]) {
        ans += s[i] + to_string(_inter[i]-i);
        sum += _inter[i]-i;
    }
    // cout << ans << endl;
    fprintf(fopen("output.txt", "w"), "%s", ans.c_str());
    int length_ans = ans.length();
    printf("Length: %d\nCompressed Length: %d\nCompression Achievement: %f\n", sum,
}
```

```
128 : 1
Length: 1000
Compressed Length: 785
Compression Achievement: 2.272611
```