PUNE INSTITUTE OF COMPUTER TECHNOLOGY
DHANKAWADI, PUNE

HIGH PERFORMANCE COMPUTING MINI-PROJECT REPORT
ON

"RUN LENGTH ENCODING"

SUBMITTED BY

Atharva Shastri	41161
Prathamesh Thombre	41172
Prathamesh Sonawane	41166

Under the guidance of
Prof. Parag Jambhulkar

DEPARTMENT OF COMPUTER ENGINEERING
Academic Year 2021-22

Contents

1 Problem Statement

Perform Run length encoding concurrently on many core GPU. Run-length encoding is a form of lossless data compression in which runs of data (sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run. GPU.

2 Abstract

Data is being generated at very high velocity in today's world.So ,data compression comes in handy a lot of times.We are implementing run length encoding in this project.R un-length encoding (RLE) is a very simple form of lossless data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run.This is most useful on data that contains many such runs. Consider, for example, simple graphic images such as icons, line drawings, and animations. It is not useful with files that don't have many runs as it could greatly increase the file size.Also,running compression parallely will save great amount of time while dealing with large data files.

3 Hardware and Software Requirements

3.1 Hardware Requirements

1. 500 GB HDD
2. 4GB RAM
3. Monitor
4. Keyboard

3.2 Software Requirements

1. 64 bit Open Source Operating System like Ubuntu 18.04
2. Jupyter Notebook

4 INTRODUCTION

- In the Run Length Encoding, we need to compute the indexes of the elements to be stored and their codes i.e. the length of the symbol run.
- The first approach for computing the codes is based on the use of another parallel primitive, the reduction, for counting the symbol or summing up the number of times a symbol appeared in its run.
- Further analysis of dependencies in the original algorithm resulted in a method which does not depend on the architectural support for the atomic operations.
- Instead of summing the number of occurrences for each symbol in parallel, the indexes of last elements in each of the run are determined on the basis of the flags.
- These index values then can be used to compute the total number of elements that appear in between these locations without actually counting the occurrences.
- The resulting count corresponds to the number of times an element appeared in its run.

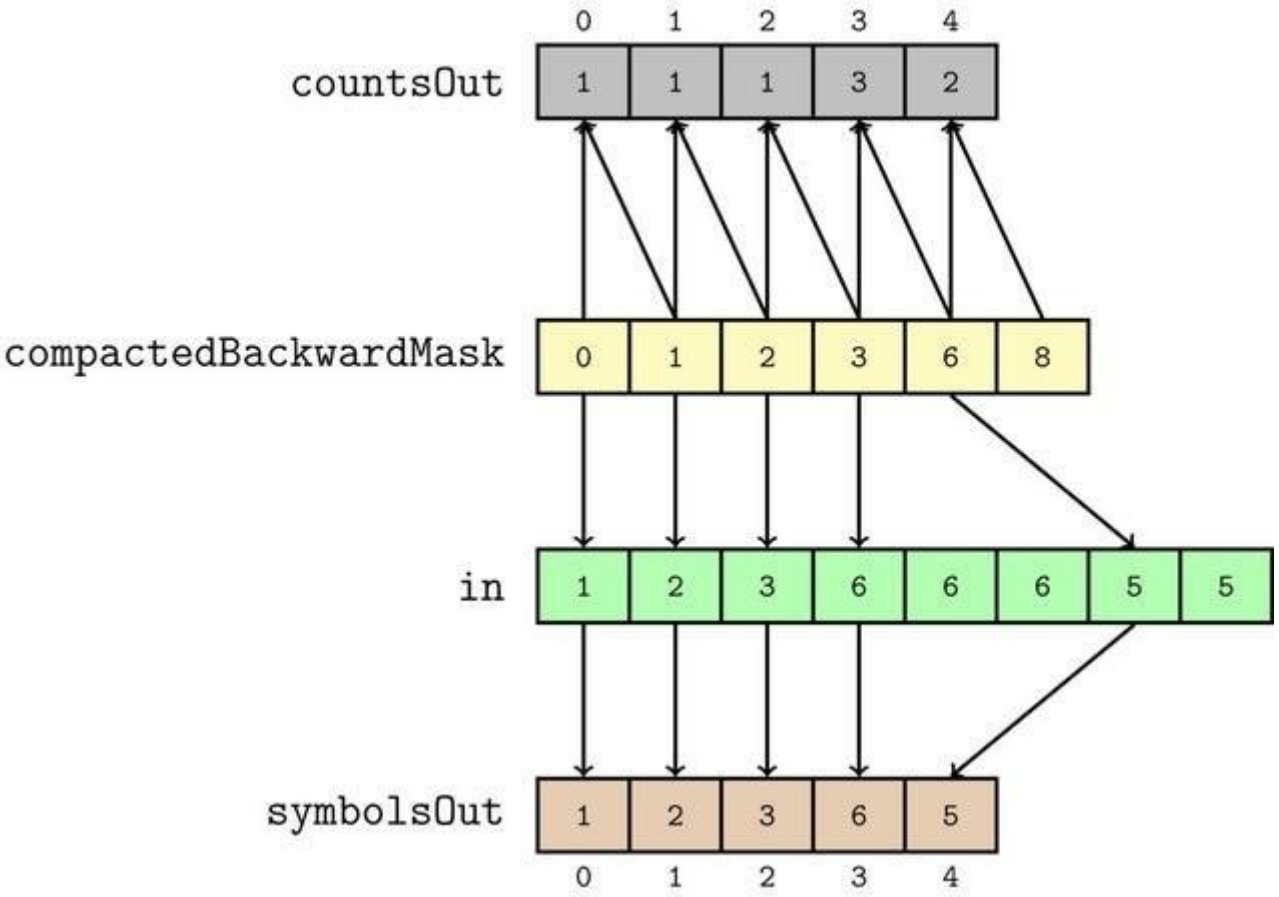
5 OBJECTIVE

- To implement a generic compression algorithm using parallelism.
- To compare the speed-up and efficiency of serial and parallel implementation of run-length encoding compression algorithm.

6 Scope

This parallel compression algorithm can be easily combined with the already available GPU-accelerated image transforms and motion estimation algorithms, to enable execution of all codec components directly on GPUs. With this approach we speed-up the entropy coding by 5-20x, and reduce the data transfer time from the GPU to system memory.

7 Methodology



8 Algorithm

```
%%cu

#include <iostream>

#include <cstring>

#include <fstream>

using namespace std;


__global__void
RunLengthEncodingComputation
(char *orig, int *_encost, 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;

      _encost[t_] = i;

   } }
```

```
// Main Function
int main()
   { int
   n=400;
   char *s = new char[n];
   int i = 0;
   string input;
   ifstream MyReadFile("input.txt");
   getline (MyReadFile, input);
   strcpy(s,input.c_str()); s[400]='\0';
   char *cudas;
   int *_encost, *_inter = new int[n];

   int threads = 4;
   int blocks = 1;
   cudaMalloc (&cudas, n*sizeof(char));
   cudaMalloc (&_encost, n*sizeof(int));

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

   RunLengthEncodingComputation <<<blocks, threads>>> (cudas, _encost, n);
   cudaDeviceSynchronize();
   cudaMemcpy(_inter, _encost, 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<<"Output:\n";
   cout << ans << endl;
}
```

9 Conclusion

Thus ,we have successfully implemented run length encoding algorithm concurrently on many core GPU