PUNE INSTITUTE OF COMPUTER TECHNOLOGY,
DHANKAWADI PUNE-43.

A
HPC Project
Report On

Generic Compression

SUBMITTED BY

Omkar Amilkanthwar	41403
Aniruddha Deshmukh	41405
Atharva Satpute	41409

CLASS: BE4

GUIDED BY

Prof. Bhumesh Masram

PICT, Department of Computer Engineering, 2021-22

ABSTRACT

In today's environment, data is created at a breakneck speed. As a result, data compression is quite useful. In this project, we're using run length encoding. Run-length encoding (RLE) is a very simple method of lossless data compression in which data runs (that is, sequences in which the same data value appears in many consecutive data elements) are saved as a single data value and count instead of the original run. This is especially handy when dealing with data that comprises a large number of such runs.

Simple visual graphics, such as icons, line drawings, and animations, are good examples. It's not recommended for files with few runs because it might significantly increase the file size. Additionally, when dealing with huge data sets, parallel compression will save a significant amount of time.

INDEX

1.PROBLEM STATEMENT

2.DOMAIN

3.METHODOLOGY

4.SERIAL ALGORITHM

5.PARALLEL ALGORITHM

6.CONCLUSION

1. PROBLEM STATEMENT

To create a parallel approach to conduct 'run length encoding' on several core GPUs at the same time.

RLE is **a basic form of data compression that converts consecutive identical values into a code consisting of the character and the number marking the length of the run**. The more similar values there are, the more values can be compressed.

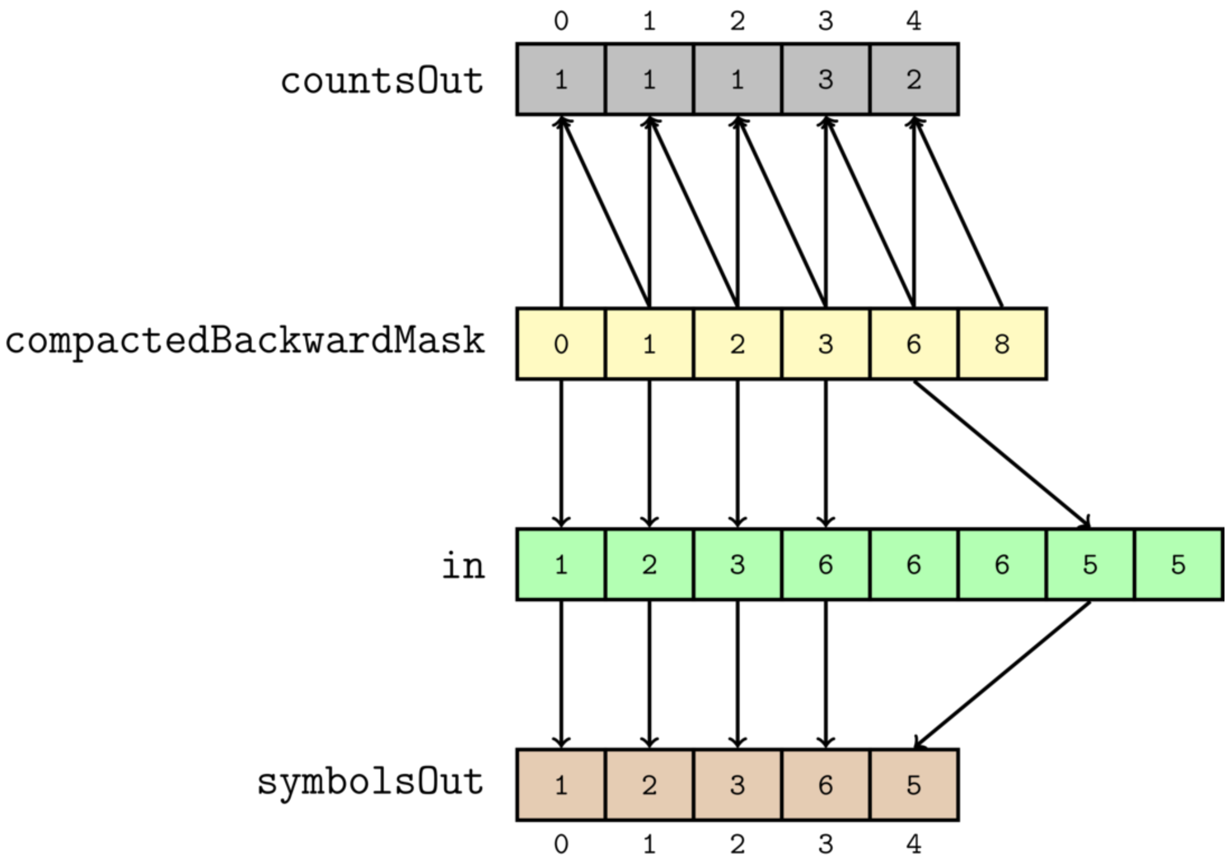
Run-length encoding (RLE) is a form of lossless data compression in which runs of data (sequences in which the same data value occurs in many consecutive data elements) are **stored as a single data value and count**, rather than as the original run. This is most useful on data that contains many such runs.

2. DOMAIN

High Performance Computing.

We'll write a technique to execute run length encoding on many core GPUs concurrently.

3.METHODOLOGY



4.SERIAL ALGORITHM

```
class RunLengthEncoding
{
   // Perform Run Length Encoding (RLE) data compression algorithm
   // on String str
   public static String encode(String str)
   {
      // stores output String
      String encoding = "";
      int count;

      for (int i = 0; i < str.length(); i++)
      {
         // count occurrences of character at index i
         count = 1;
         while (i + 1 < str.length() && str.charAt(i) == str.charAt(i+1)) {
            count++;
            i++;
         }

         // append current character and its count to the result
         encoding += String.valueOf(count) + str.charAt(i);
      }

      return encoding;
   }

 }
```

5.PARALLEL ALGORITHM

// Backward Masks of all elements are calculated with this program.

```
__global__void maskKernel(int *g_in, int* g_backwardMask, int n) {
    for (int i : hemi::grid_stride_range(0, n)) {

        if (i == 0)
            g_backwardMask[i] =
            1;

        else {
            g_backwardMask[i] = (g_in[i] != g_in[i - 1]);
        }

    }
}



__global__void compactKernel(int* g_scannedBackwardMask,
                             int* g_compactedBackwardMask,
                             int* g_totalRuns,
                             int n) {

    for (int i : hemi::grid_stride_range(0, n)) {

        if (i == (n - 1)) {
            g_compactedBackwardMask[g_scannedBackwardMask[i]]
            = i + 1;
            *g_totalRuns = g_scannedBackwardMask[i];
        }

        if (i == 0) {
            g_compactedBackwardMask[0]
```

```
        = 0;
      }
      else if (g_scannedBackwardMask[i] !=
        g_scannedBackwardMask[i - 1]) {
        g_compactedBackwardMask[g_scannedBackwardMask[i] - 1]
        = i;
      }
    }
```

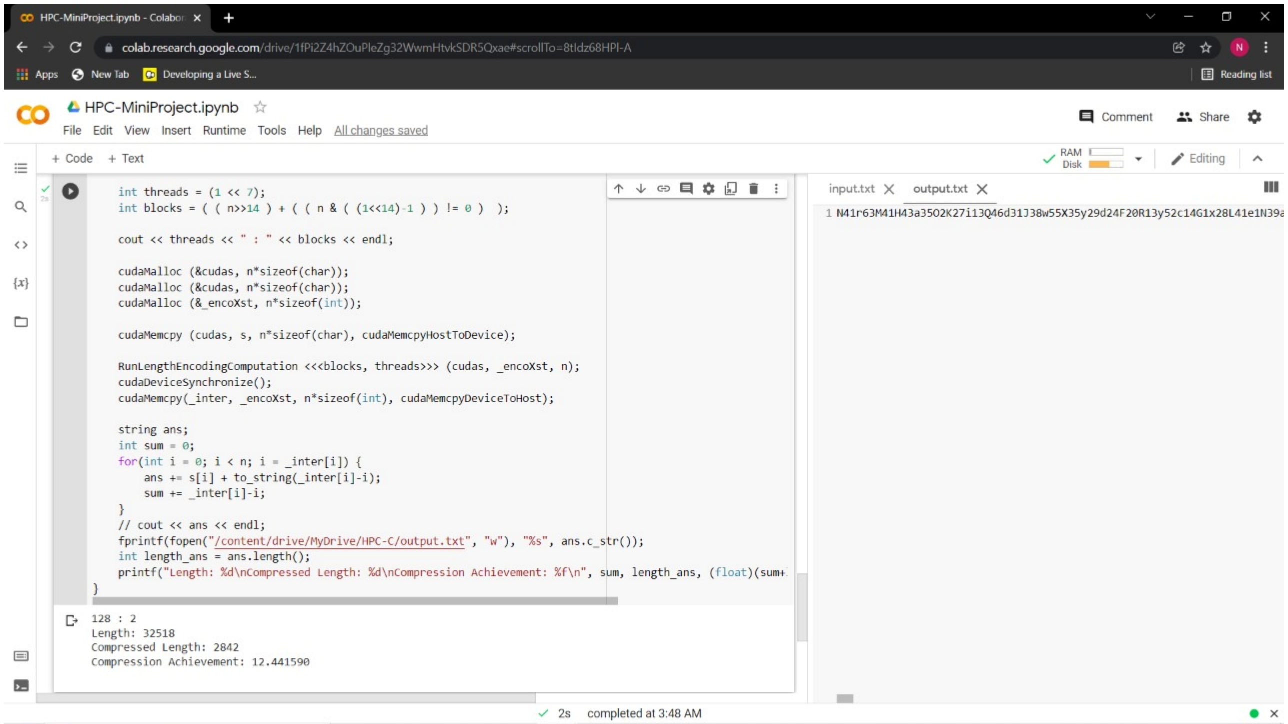
//Final Compression Algorithm

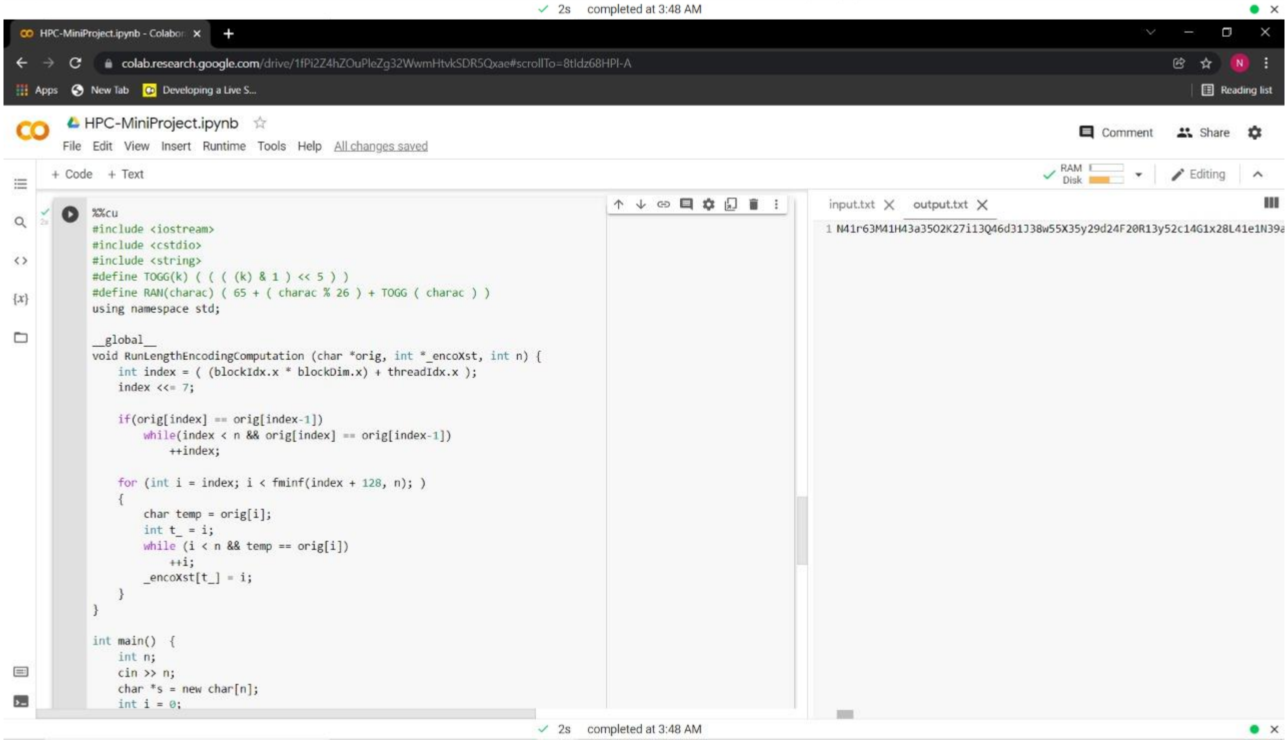
```
__global__void scatterKernel(
       int* g_compactedBackwardMask,
     int* g_totalRuns,
     int* g_in,
     int* g_symbolsOut,
     int* g_countsOut) {
  int n = *g_totalRuns;

 for (int i : hemi::grid_stride_range(0, n)) {
   int a = g_compactedBackwardMask[i];
   int b = g_compactedBackwardMask[i + 1];

     g_symbolsOut[i] =
     g_in[a]; g_countsOut[i]
     = b - a;
   }
 }
```

6.Output


HPC-MiniProject.ipynb - Colabor
colab.research.google.com/drive/1fPi2Z4hZOuPleZg32WwmHtvkSDR5Qxae#scrollTo=8tldz68HPI-A
HPC-MiniProject.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Editing
input.txt output.txt
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("/content/drive/MyDrive/HPC-C/output.txt", "w"), "%s", ans.c_str());
int length_ans = ans.length();
printf("Length: %d\nCompressed Length: %d\nCompression Achievement: %f\n", sum, length_ans, (float)(sum+
}
128 : 2
Length: 32518
Compressed Length: 2842
Compression Achievement: 12.441590
1 N41r63M41H43a3502K27i13Q46d31J38w55X35y29d24F20R13y52c14G1x28L41e1N39a
2s completed at 3:48 AM


HPC-MiniProject.ipynb - Colabor
colab.research.google.com/drive/1fPi2Z4hZOuPleZg32WwmHtvkSDR5Qxae#scrollTo=8tldz68HPI-A
HPC-MiniProject.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Editing
input.txt output.txt
%%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;
cin >> n;
char *s = new char[n];
int i = 0;
1 N41r63M41H43a3502K27i13Q46d31J38w55X35y29d24F20R13y52c14G1x28L41e1N39a
2s completed at 3:48 AM

7.CONCLUSION

As a result, we were able to successfully implement the run length encoding technque on a multi-core GPU.