PUNE INSTITUTE OF COMPUTER TECHNOLOGY,
DHANKAWADI PUNE-43.

A Project Report on

Generic Compression

SUBMITTED BY

NAME: Chetan Atole (41307)

NAME: Prem Bansod (41310)

NAME : Shailesh Borate (41315)

CLASS: BE3

GUIDED BY

Prof. S. H. Pisey

COMPUTER ENGINEERING DEPARTMENT

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.Run-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.

CONTENTS

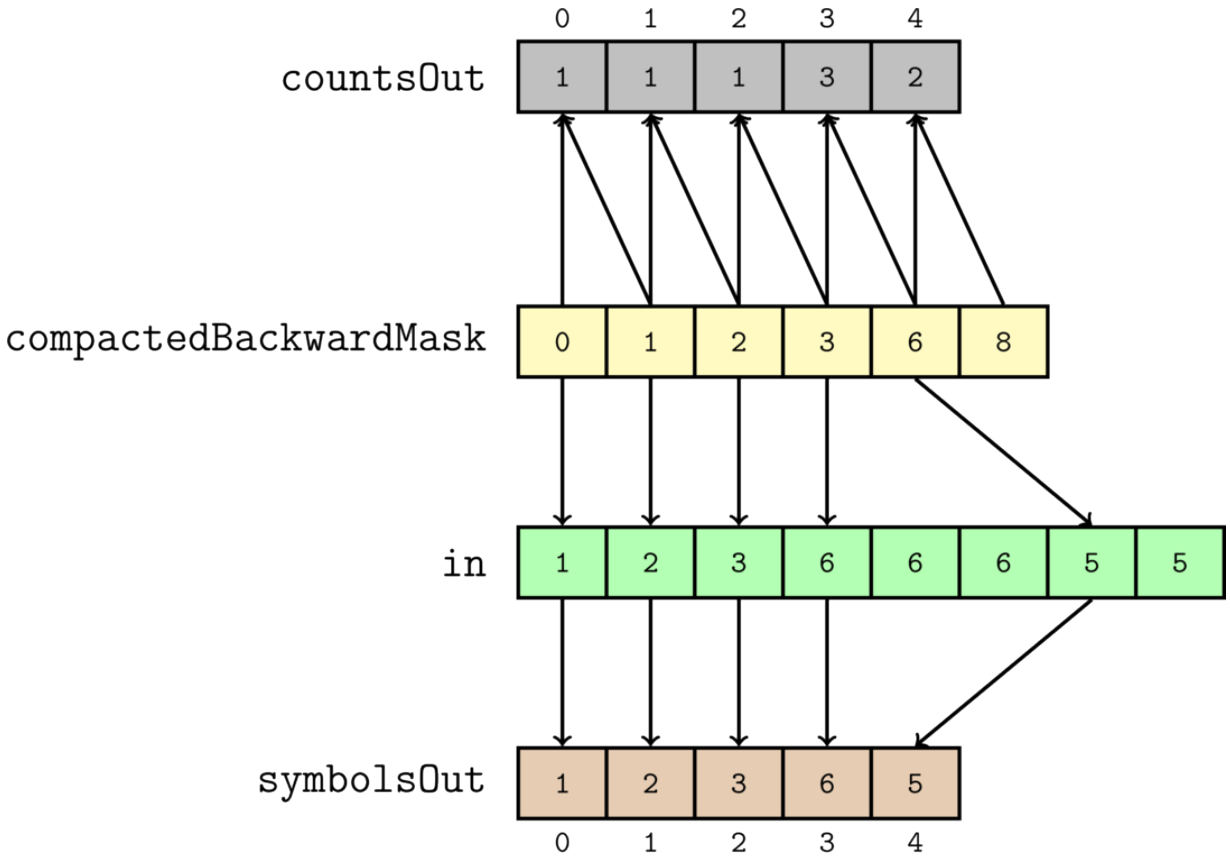
1. PROBLEM STATEMENT

Write a parallel algorithm to run ‘run length encoding’ concurrently on many core GPU.

2. DOMAIN

High Performance Computing.
We will be writing an algorithm to run run length encoding concurrently on many core GPU.

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

//Program to calculate Backward Masks of all elements

```
__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.CONCLUSION

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