

**PUNE INSTITUTE OF COMPUTER TECHNOLOGY  
DHANKAWADI, PUNE**

**HIGH PERFORMANCE COMPUTING MINI-PROJECT REPORT  
ON**

**“GENERIC COMPRESSION - RUN LENGTH ENCODING  
CONCURRENTLY ON MULTI-CORE.”**

**SUBMITTED BY**

Maitraya Kakade 41427

Pranav Kulkarni 41430

**Under the guidance of**  
Prof. Deepali Kadam



**DEPARTMENT OF COMPUTER ENGINEERING  
Academic Year 2020-21**

---

## Contents

<b>1</b>	<b>Problem Statement</b>	<b>1</b>
<b>2</b>	<b>Abstract</b>	<b>2</b>
<b>3</b>	<b>INTRODUCTION</b>	<b>3</b>
<b>4</b>	<b>OBJECTIVE</b>	<b>4</b>
<b>5</b>	<b>Scope</b>	<b>5</b>
<b>6</b>	<b>System Architecture</b>	<b>6</b>
<b>7</b>	<b>Test Cases</b>	<b>7</b>
<b>8</b>	<b>Result</b>	<b>8</b>
<b>9</b>	<b>Conclusion</b>	<b>9</b>
	<b>References</b>	<b>10</b>

---

# 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.

---

## 2 Abstract

Multi-core programming is increasingly used for acceleration of data-parallel functions, such as image transforms and motion estimation. However, the entropy coding stage is typically executed on the CPU due to inherent data dependencies in lossless compression algorithms. We propose a simple way of efficiently dealing with these dependencies, and present a parallel compression algorithm specifically designed for efficient execution on many-core architectures. By parallelization of lossless compression algorithms not only do we significantly speed-up the entropy coding, but we also enable completion of all encoding phases using OpenMPI.

---

### 3 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.

---

## 4 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.

---

## 5 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.

---

## 6 System Architecture

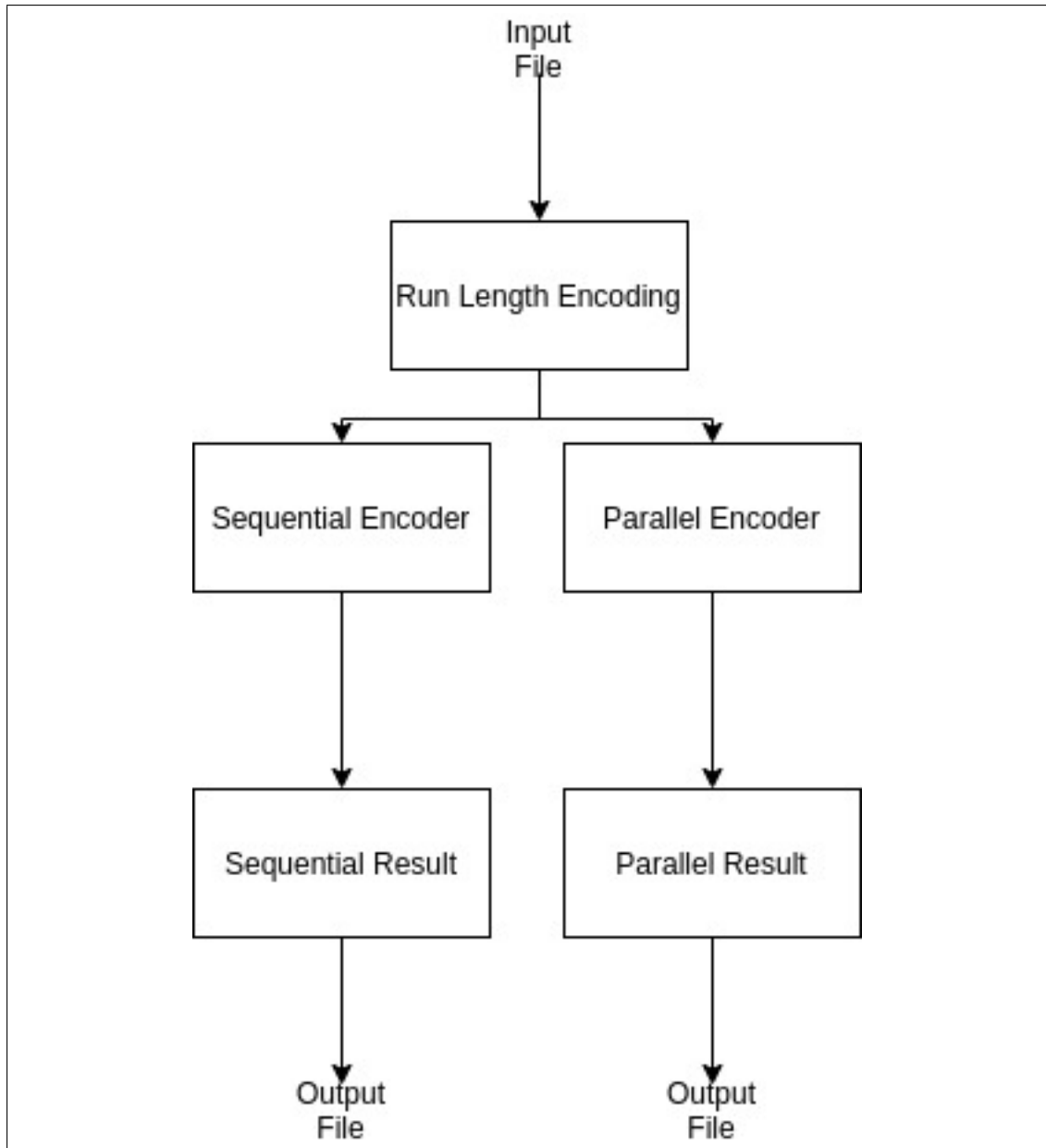


Figure 1: System Architecture



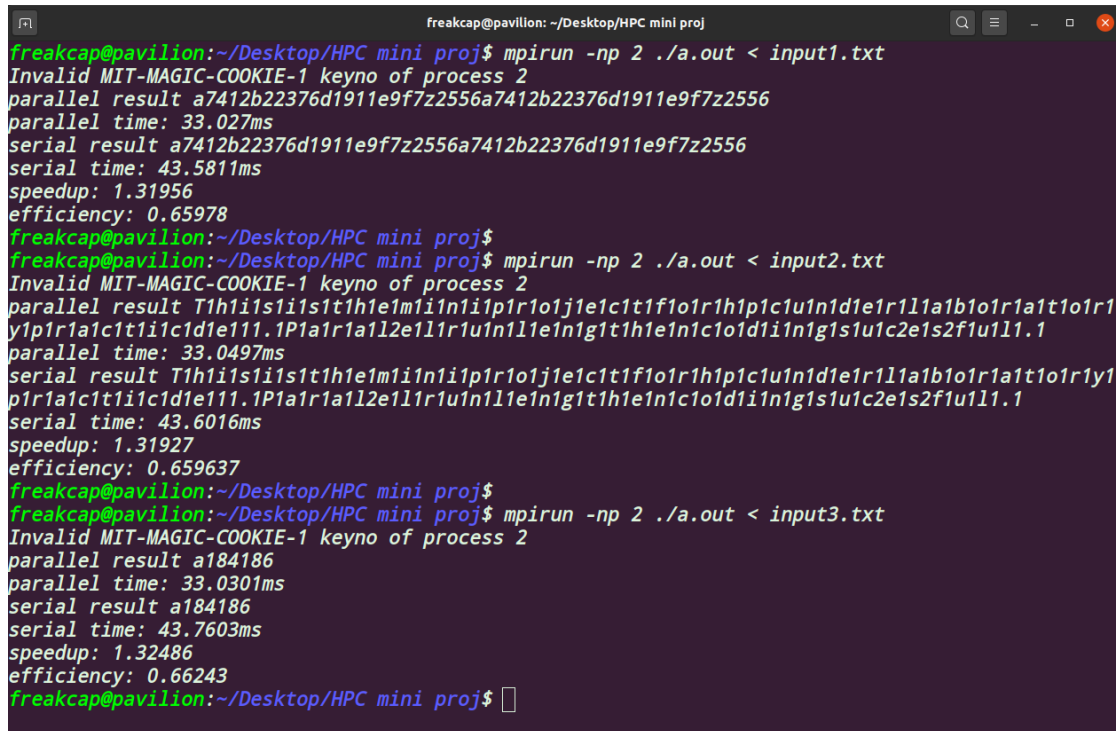
## 7 Test Cases

Files :

input1 => average case

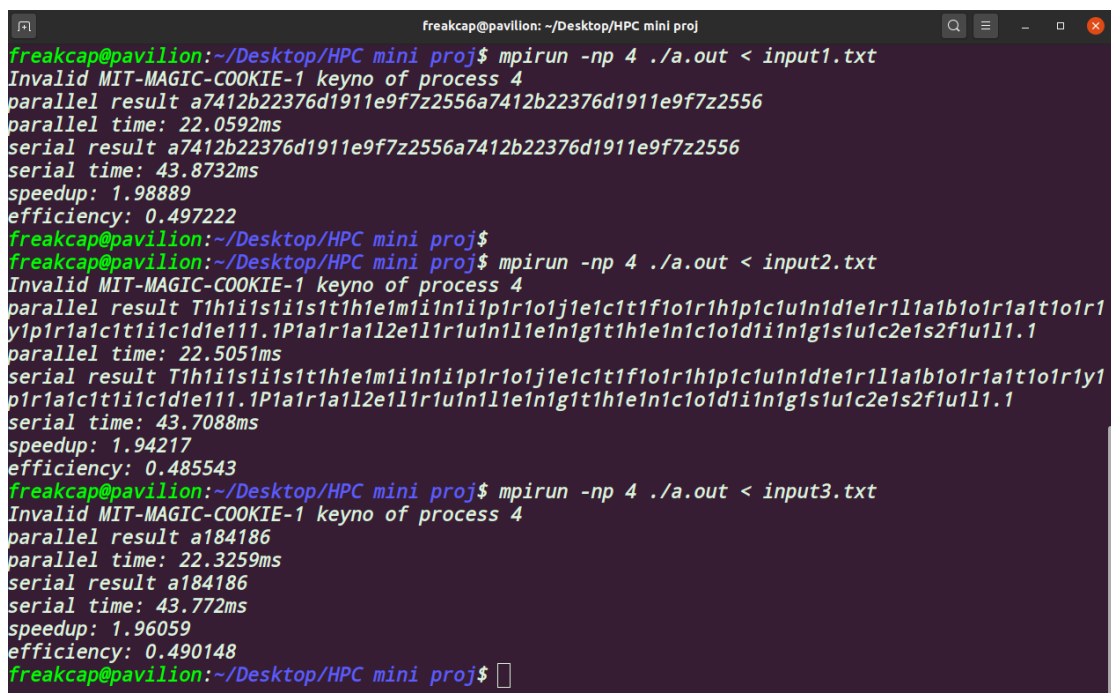
input2 => worst case

input3 => best case

A terminal window titled 'freakcap@pavilion: ~/Desktop/HPC mini proj' showing the execution of a program with 2 processes. The user runs 'mpirun -np 2 ./a.out < input1.txt'. The output shows a parallel result, parallel time (33.027ms), serial result, serial time (43.581ms), speedup (1.31956), and efficiency (0.65978). The user then runs 'mpirun -np 2 ./a.out < input2.txt' and 'mpirun -np 2 ./a.out < input3.txt', both showing similar performance metrics with slight variations in time and efficiency.

```
freakcap@pavilion:~/Desktop/HPC mini proj$ mpirun -np 2 ./a.out < input1.txt
Invalid MIT-MAGIC-COOKIE-1 keyno of process 2
parallel result a7412b22376d1911e9f7z2556a7412b22376d1911e9f7z2556
parallel time: 33.027ms
serial result a7412b22376d1911e9f7z2556a7412b22376d1911e9f7z2556
serial time: 43.581ms
speedup: 1.31956
efficiency: 0.65978
freakcap@pavilion:~/Desktop/HPC mini proj$
freakcap@pavilion:~/Desktop/HPC mini proj$ mpirun -np 2 ./a.out < input2.txt
Invalid MIT-MAGIC-COOKIE-1 keyno of process 2
parallel result T1h1i1s1i1s1t1h1e1m1i1n1i1p1r1o1j1e1c1t1f1o1r1h1p1c1u1n1d1e1r1l1a1b1o1r1a1t1o1r1y1p1r1a1c1t1i1c1d1e111.1P1a1r1a1l2e1l1r1u1n1l1e1n1g1t1h1e1n1c1o1d1i1n1g1s1u1c2e1s2f1u1l1.1
parallel time: 33.0497ms
serial result T1h1i1s1i1s1t1h1e1m1i1n1i1p1r1o1j1e1c1t1f1o1r1h1p1c1u1n1d1e1r1l1a1b1o1r1a1t1o1r1y1p1r1a1c1t1i1c1d1e111.1P1a1r1a1l2e1l1r1u1n1l1e1n1g1t1h1e1n1c1o1d1i1n1g1s1u1c2e1s2f1u1l1.1
serial time: 43.6016ms
speedup: 1.31927
efficiency: 0.659637
freakcap@pavilion:~/Desktop/HPC mini proj$
freakcap@pavilion:~/Desktop/HPC mini proj$ mpirun -np 2 ./a.out < input3.txt
Invalid MIT-MAGIC-COOKIE-1 keyno of process 2
parallel result a184186
parallel time: 33.0301ms
serial result a184186
serial time: 43.7603ms
speedup: 1.32486
efficiency: 0.66243
freakcap@pavilion:~/Desktop/HPC mini proj$
```

Figure 2: Output with no. of processes 2

A terminal window titled 'freakcap@pavilion: ~/Desktop/HPC mini proj' showing the execution of a program with 4 processes. The user runs 'mpirun -np 4 ./a.out < input1.txt'. The output shows a parallel result, parallel time (22.0592ms), serial result, serial time (43.8732ms), speedup (1.98889), and efficiency (0.497222). The user then runs 'mpirun -np 4 ./a.out < input2.txt' and 'mpirun -np 4 ./a.out < input3.txt', both showing similar performance metrics with slight variations in time and efficiency.

```
freakcap@pavilion:~/Desktop/HPC mini proj$ mpirun -np 4 ./a.out < input1.txt
Invalid MIT-MAGIC-COOKIE-1 keyno of process 4
parallel result a7412b22376d1911e9f7z2556a7412b22376d1911e9f7z2556
parallel time: 22.0592ms
serial result a7412b22376d1911e9f7z2556a7412b22376d1911e9f7z2556
serial time: 43.8732ms
speedup: 1.98889
efficiency: 0.497222
freakcap@pavilion:~/Desktop/HPC mini proj$
freakcap@pavilion:~/Desktop/HPC mini proj$ mpirun -np 4 ./a.out < input2.txt
Invalid MIT-MAGIC-COOKIE-1 keyno of process 4
parallel result T1h1i1s1i1s1t1h1e1m1i1n1i1p1r1o1j1e1c1t1f1o1r1h1p1c1u1n1d1e1r1l1a1b1o1r1a1t1o1r1y1p1r1a1c1t1i1c1d1e111.1P1a1r1a1l2e1l1r1u1n1l1e1n1g1t1h1e1n1c1o1d1i1n1g1s1u1c2e1s2f1u1l1.1
parallel time: 22.5051ms
serial result T1h1i1s1i1s1t1h1e1m1i1n1i1p1r1o1j1e1c1t1f1o1r1h1p1c1u1n1d1e1r1l1a1b1o1r1a1t1o1r1y1p1r1a1c1t1i1c1d1e111.1P1a1r1a1l2e1l1r1u1n1l1e1n1g1t1h1e1n1c1o1d1i1n1g1s1u1c2e1s2f1u1l1.1
serial time: 43.7088ms
speedup: 1.94217
efficiency: 0.485543
freakcap@pavilion:~/Desktop/HPC mini proj$
freakcap@pavilion:~/Desktop/HPC mini proj$ mpirun -np 4 ./a.out < input3.txt
Invalid MIT-MAGIC-COOKIE-1 keyno of process 4
parallel result a184186
parallel time: 22.3259ms
serial result a184186
serial time: 43.772ms
speedup: 1.96059
efficiency: 0.490148
freakcap@pavilion:~/Desktop/HPC mini proj$
```

Figure 3: Output with no. of processes 4

---

## **8 Result**

Highest speedup and efficiency is given when 4 process are used for all i.e best case, worst case and average case.

---

## 9 Conclusion

We presented parallel lossless compression designed for efficient execution on many core architectures supporting parallelism. The parallelization of the Run-Length encoding resulted in processing rates up to 2x speedup.

---

## References

[1] <https://www.geeksforgeeks.org/run-length-encoding/>

[2] <https://stackoverflow.com/questions/31890523/how-to-use-mpi-gathe>

[3] <https://www.open-mpi.org/doc/v4.0/>