University of Warsaw
Faculty of Mathematics, Informatics and Mechanics

Adam Deryło
Student no. 432952

Adrian Hess
Student no. 431481

Magdalena Pałkus
Student no. 421537

Michał Skwarek
Student no. 418426

GPU acceleration of CCSDS Rice decoding

Bachelor's thesis
in COMPUTER SCIENCE

Supervisor:
Paweł Gora
Faculty of Mathematics, Informatics and Mechanics, University of Warsaw

Warsaw, Jan 2023

Abstract

TBD

Keywords

CCSDS Rice coding, GPU, Compute Unified Device Architecture (CUDA)

Thesis domain (Socrates-Erasmus subject area codes)

11.3 Informatics, Computer Science

Subject classification

D. Software
D.1.3. Concurrent Programming
I.4.2. Compression (Coding)

Tytuł pracy w języku polskim

Akcerleracja GPU dekodowania CCSDS Rice

Contents

Chapter 1

Introduction

1.1. Background and motivation

Machine learning (ML) has rapidly become a crucial tool for solving complex problems in a variety of domains, including computer vision, natural language processing, and speech recognition. However, ML pipelines can often experience significant bottlenecks due to multi-stage data processing pipelines that include loading, decoding, cropping, resizing, and other augmentations. This is particularly true when these processing steps are executed on a CPU, which is frequently the case. Since, most of the machine learning training is already GPU accelerated due to ML workloads having a highly parallel nature, it is a natural progression to solve data processing bottlenecks by harnessing GPU acceleration.

One such bottleneck is decompression of RICE coded data, which is one of the most commonly used compression algorithms used in astronomy and astrophysics. Currently, there are only CPU-based solutions for decoding RICE compressed data. This, substantially hinders application of machine learning solutions in astronomy, especially considering the massive amount of training data already stored in RICE compressed algorithm as well as being produced by observatories all around the world. To illustrate, just Solar Dynamics Observatory (SDO), with limited bandwidth due to being a satellite, is producing 130 megabits of rice coded data per second, that is 500 TB per year.

1.2. Problem statement

The problem at hand is that the current RICE decompression algorithm is a computationally intensive and cannot keep up with the demand for high throughput and real-time processing. The goal of this project is to design and develop a GPU-based RICE decompression algorithm that can significantly accelerate the decompression process and improve the speed, efficiency, and scalability of ML pipelines in astronomy applications.

1.3. Research objectives

The main objectives of this thesis are as follows:

- To study the existing literature on GPU acceleration of data compression and RICE decompression.
- To analyze the performance of RICE decompression on a CPU and a GPU.

- To design and implement a GPU-accelerated RICE decompression algorithm.
- To evaluate the performance of the GPU-accelerated RICE decompression algorithm and compare it with the CPU-based RICE decompression algorithm.
- And finally, to incorporate findings into NVIDIA Data Loading Library (DALI), so machine learning as well as the astronomy community can utilize our findings.

1.4. Thesis structure

In this thesis, we present our attempts at parallelizing the RICE decompression algorithm and CUDA-based implementations of our findings.

Chapter 2 aims to define and explain key terms for our problem space, including the RICE algorithm itself. In the following chapter, we review previous work on the subject. We then present our solution architecture and how it ties into NVIDIA's open-source data loading framework called DALI. Additionally, we introduce the Compute Unified Device Architecture (CUDA) to the reader. In Chapter 5, we propose a parallel implementation of the algorithm on GPU with details. Chapter 6 presents our performance and speedup results. Finally, in Chapter 7, we summarize our findings and conclude the thesis.

Chapter 2

Key terms

2.1. RICE

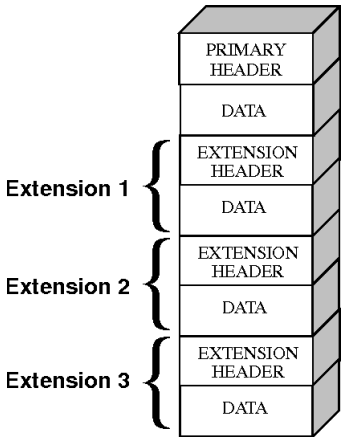
The RICE algorithm is a well-known, lossless data compression algorithm, that uses a set of variable-length codes to compress data. The essence of this algorithm is using these codes to change symbols that are expected to be more frequent to shorter code words. The algorithm works on data blocks that are encoded independently, it's not needed to transfer some information between different packets. It improves performance and makes RICE algorithm performance independent of data packet size. The RICE algorithm consists of two main parts: a preprocessor and an adaptive entropy coder.

2.2. FITS

The Flexible Image Transport System (FITS) format is a digital file format that is commonly utilized in the field of astronomy for storing and transferring scientific data. FITS files typically contain images, data cubes, and tables of observational data, as well as associated metadata. One of the key characteristics of FITS is its ability to store multiple data arrays in a single file, which allows for the efficient storage and transfer of large data sets. A FITS files are composed of the following FITS structures, in the order listed:

- Primary header and data unit (HDU).
- Conforming Extensions (optional).
- Other special records (optional, restricted).

Thus, files usually resemble the following schema:



Where, all headers, including the primary one, contain relevant metadata as a list of keys and value pairs. Furthermore, according to the most recent FITS standard published by NASA, there are three types of standard extensions:

- IMAGE extensions.
- TABLE ASCII-table extensions; and
- BINTABLE binary-table extensions

Chapter 3

Previous work

3.1. Rice Coding Suitable for GPU Massively Parallel Implementation

3.2. Current decompresssion solutions

Chapter 4

Context

4.1. NVIDIA

4.2. GPU Programming and CUDA

4.3. DALI

Chapter 5

Solution architecture

5.1. DALI readers

5.2. CUDA

Chapter 6

GPU acceleration of RICE decoding

6.1. Naive approach

6.2. Optimizing asynchronous data transfer

Chapter 7

Results and analysis

7.1. Performance evaluation metrics

7.2. Benchmarking suite

7.3. Comparison of various approaches

Chapter 8

Conclusion

8.1. Summary of findings

8.2. Contributions

8.3. Future work

Bibliography

[Bea65] Juliusz Beaman, *Morbidity of the Jolly function*, Mathematica Absurdica, 117 (1965) 338–9.