Compression Using Massively Parallel Processors

By James Rodiger Advisor Dr. Takunari Miyazaki

Data Compression

By identifying repeated patterns of bytes the amount of space required to store data can be greatly reduced.
Currently there are many different ways of compressing data but very few have been made to run on massively parallel processors

Project Goals

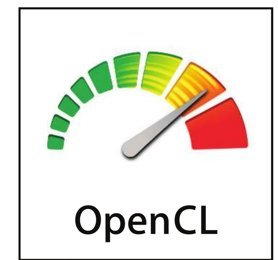
The goal of this project is to create a working implementation of the DEFLATE compression algorithm in Opencl to test the efficiency and effectiveness of massively parallel processing for compression. This project will also be licensed under the GNU GPL so that it is free software and can be used by others without restrictions.

Massively Parallel Processing

CPU's only have two to four cores usually which means only a few points of data can be processed simultaneously

GPU's have many cores running at a lower clock speed but can have far greater throughput of data.

By using Opencl the DEFLATE compression algorithm can be run with increased efficiency on large file sizes.



The DEFLATE Specification

Defined by RFC-1951 it is a combination of LZ77 + Huffman Coding

LZ77 identifies patterns in the data and reduces them to length and distance pairs

Works in a sliding window of 32kb to back reference and generate length distance pairs

Data is divided into blocks of three types

00 - Uncompressed
01 - Compressed with a fixed Huffman tree
10 - Compressed with a dynamic Huffman tree

Huffman Coding identifies the most frequently used bytes and reduces the number of bits needed to represent them

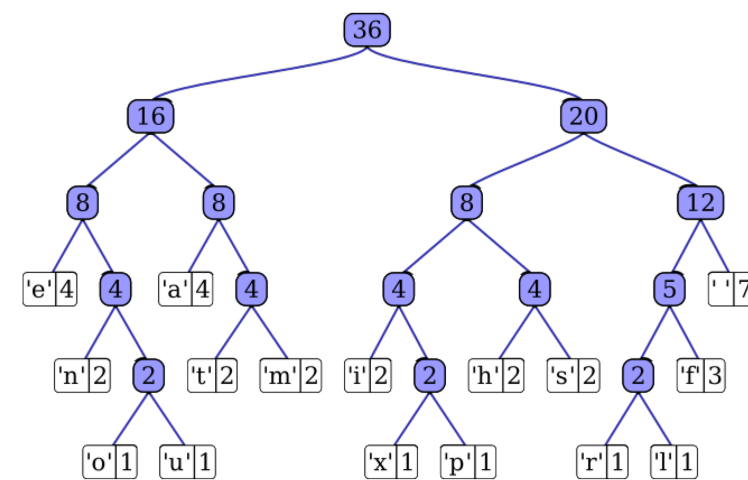


Image Source Wikipedia

Efficiency of Massively Parallel Compression

Data compression is inherently sequential and difficult to parallelize and this sentiment is represented in the results of my project. The main parallelizeable element of DEFLATE is the LZ77 algorithm. The pattern matching can be done more efficiently with many cores but the time it takes to copy the data to and from the GPU is long and must be done often with large files. This causes the efficiency of the algorithm to drop significantly. Given more time I believe that my implementation could be optimized by using parallel processing on the CPU as well as the GPU to allow for simultaneous loading of data to and from the GPU memory and either writing to the disk or generating Huffman codes to greatly increase efficiency.