15-418 Project Proposal: LZ Compression

Jack Woodson (jrwoodso), Alex Li (alexli3)

URL: https://jrwoodso.github.io/15-418-LZ_Compression/

Summary

Will investigate how to parallelize the serial compression algorithms LZ77 and LZW. As we try to parallelize, we would also compare how much performance (compression ratio) we can retain while speeding up the computation. We will target the GHC machine, and extend to the PSC to test scalability on higher core counts.

Background of LZ Compression

LZ-family compression algorithms form the foundation of widely-used formats like ZIP, PNG, and GIF. Both algorithms exploit repetition in data: rather than storing repeated sequences in full, they encode references to prior occurrences, achieving significant size reduction.

There are various different forms of LZ compression and the methods that either benefit speed with low compression ratio, vice versa, or memory consumption. The different LZ compression algorithms that we will be increasing their performance are LZ77 and LZW. The LZ77 makes use of a sliding windows technique to process current data with previously processed / compressed data. The LZW algorithm will take in a set of symbols and convert them into a code that takes less file size. These codes and symbols are stored in a dictionary for easy, low cost lookup. If these symbols appear again they are substituted for the short code.

Challenge

These algorithms heavily rely on finding code or symbols that are repeated early on in the compression process to be able to shorten the repetition. When working with multiple threads this task becomes increasingly more difficult as they will either create unique codes for these symbols individually taking up too much memory leading to a poor compression ratio, or waiting for threads to load to memory creating specific bottlenecks.

There are memory access characteristics to consider, communication to computation issues, divergent execution and load balancing, as well as compression ratio performance constraints. Our project will tread this fine line to be able to optimize our code working with multiple threads while maintaining a good compression ratio.

Resources

The machines that we will be conducting our code on are the GHC machines (with up to 8 cores), and PSC machines (up to 128 cores). We will emphasize the algorithm's ability to scale with higher thread counts while maintaining a good compression ratio.

As for the code we plan to write this code from scratch while making use of helpful references such as:

- LZ77 — https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-wusp/fb98aa28-5cd7-407f-8869-a6cef1ff1ccb
- LZW — https://eng.libretexts.org/Bookshelves/Electrical_Engineering/Signal_Processing_and_Modeling/Information_and_Entropy_(Penfield)/03%3A_Compression/3.07%3A_Detail-_LZW_Compression/3.7.01%3A_LZW_Algorithm_Example_1

No additional resources beyond GHC and PSC should be needed. We anticipate potential issues with scaling on the PSC that will not be noticed on the GHC machines at 8 cores.

Goals and Deliverables

Baseline Success

We plan to achieve working sequential implementations of the LZ77 and LZW algorithms as a baseline. They should perform decently and have competitive compression ratios on standard benchmarks. We also plan to achieve a parallel algorithm for LZ77. Tentatively, we think at least 4–6x speedup on 8 GHC cores with less than 15% degradation in compression ratio should be achievable.

After getting a version of LZ77 working well, we plan to extend our work to the LZW method as well as begin exploring scaling on the PSC on various input sizes. We hope to achieve good speedup without significantly worse cost metrics. All these scenarios would also be accompanied by an analysis of what limits scaling (memory, synchronization, load imbalance).

Hope to Achieve

The project is already quite aggressive; it almost already encompasses what we hope to achieve. For extensions to our project scope, we would be interested in further improving speed and performance. For example, alternative pre-initialization strategies, as well as speculative approaches could be explored.

Fallback

If the work is significantly harder to implement, or harder to parallelize, we may choose to only focus on LZ77 or try to refine the parallel LZ77 implementation instead. Or, we may only do a deeper analysis on LZ77 and then present more surface-level observations on LZW.

Platform Choice

Our platform choice is going to primarily consist of work done on the GHC machines. This is due to the availability we have as students and able to play around with different thread counts with these compression algorithms. Additionally, as we are trying to create the most parallel efficient algorithms we also plan to make use of the PSC machines to be able to test and see how our algorithm runs with higher thread counts 16–128. The PSC is beneficial in this case in which we can demonstrate just how good our algorithms are

when there are significantly more threads working on compression and how our code works with this increased computing power to be able to have a good compression ratio.

Also, a multicore CPU is also a better fit for this workload than a GPU. LZ compression involves irregular, backward-looking memory accesses (LZ77's sliding window) and dynamic dictionary lookups (LZW) — both of which map poorly to GPU architectures that require coalesced memory access patterns to perform well. Additionally, the sequential data dependencies inherent to both algorithms limit the degree of fine-grained parallelism available, making the overhead of GPU thread management difficult to justify.

Schedule

Mar 30 – Apr 4 Working LZ77 and LZW sequential algorithms that yield an optimal compression ratio.

Important to be able to have a good sequential baseline to measure our efforts in increasing speedup.

Additionally create a good test set that comprises files of different information, highly repetitive, no repetition and of different sizes.

Apr 5 – Apr 11 Working compression algorithm for LZ77 that are able to achieve relatively high speedup on lower thread counts but are in the right direction in being able to get a good workload balance and scaling at higher thread counts.

As there are similarities between LZ77 and LZW we will primarily focus on LZ77 first as a lot of the ideas for increased performance should translate to the LZW method.

Keep up to date with changes and requirements in milestone for April 14.

Apr 12 – Apr 18 Have a LZ77 compression algorithm that has good scalability and good compression ratio with some small improvements.

Keep website up to date with milestone and changes made.

Apr 19 – Apr 30 Have a LZW compression algorithm that has good scalability and good compression ratio with some small improvements. Compare these two different algorithms that we've been working to find the best compression algorithm with lower thread counts, to higher, and memory usage.

Additionally, meet all other requirements for website.

References

Just a collection of the most relevant literature we have found that we hope will help our project.

[1] https://ieeexplore.ieee.org/document/6543048

[2] https://ieeexplore.ieee.org/document/8671624

[3] https://ieeexplore.ieee.org/document/8929928

[4] https://www.cs.hiroshima-u.ac.jp/cs/_media/cp14.pdf