Accelerating Compression Algorithms using Graphics Hardware

Brandon James Talbot
Department of Computer Science
University of Cape Town

ABSTRACT

Graphics cards are capable of doing a lot more work at the same time than the CPU (Central Processing unit). This allows them to do many algorithms in shorter time than when run on the CPU. This does require the algorithm to have the possibility of being run in parallel. We will be going over some of the compression algorithms out there, and show that compression algorithms can be run on large datasets within a short time. This will then show that we will be capable of compressing large sets of floating point values in real time. I will show that compared to the CPU the algorithms could achieve a 2 - 200+ times speed up. I will also show that for large datasets we could keep the compression time below 6 seconds making it feasible for streaming data.

1. INTRODUCTION

Compression algorithms are used to make the size of a computer file smaller. There are many compression algorithms out there: Some that are designed specifically to work on a certain type of file and others that work on any type of file. The most commonly used used compression file formats are: RAR, ZIP, JPEG, MP3, GIF and MPEG.
Using algorithms such as: Huffman[4], LZO[3], LZSS[6], DXT[1] etc.

Compression algorithms also fall into another 2 sections: Lossless and lossy compression.
Lossy compression is a type of compression that could cause a slight loss of data. For instance in JPEG compression the algorithm removes data that it determines is not pertinent to the display of the image. The 3 colour ranges (red, green and blue) are kept but many of the alpha values are taken out. Also after the EOF (End of file) some image formats save data, but when sent through the JPEG compression algorithm it is all removed. Another type of lossy compression is one that compresses each bit as much as possible, that the decompressing algorithm sometimes does not receive the exact same values back.
Lossless data compression is a compression algorithm that does not remove any Data from the file and whose decompression algorithm will always retrieve the exact same file back.

Most compression algorithms are linear, they compress the file by staring at a beginning point and ending at a end point. Nevertheless most of these algorithms can be Parallelized. The easiest form to convert the algorithm into a parallel version is to break the algorithm into "Blocks" and each block could be compressed at the same time and then be placed together when all have finished compressing. Thus speeding up the compression algorithm tremendously.
Some algorithms are difficult to parallelize due to interdependencies. For example: The compression involves creating sequential data, but if two threads try to place data at the same time they will write the same number rather than increment.
Most current parallel algorithms are simple PThreaded(CPU threaded) algorithms. Many of which could be converted into GPU (Graphical Processing unit) based algorithms. This will allow us to compress a lot more of the file at the same time, thus theoretically shorten the time required to run the algorithm.

The speed increase in these algorithms is a great necessity. The SKA (Square Kilometer Array), a radio telescope system in South Africa, requires compression of huge floating point data files for transmission from the satellites to the offices. The files are 1TB (terabyte) in size and come in from the satellites every second. The algorithm thus needs to compress the 1TB floating point data files fast enough to not cause any backlogs.

2. GPU BASED COMPRESSION ALGORITHMS

2.1 Image Compression

Since the GPU was designed to display images and triangle meshes faster than the CPU, as well as do fast edits to them on the fly, it makes sense for the compression algorithms to be a lot faster on the GPU.

A group from the University of Toronto created a GPU based JPEG compression algorithm that was determined to be 20 times faster and due to the procedure of compression that speed up gives only around 15 times performance gain for the algorithm. This is because of the parallelism making the ratio of compression slightly smaller[8]. They also noticed that the larger the image resolution the greater speed up you achieve since GPUs are capable of doing more in the same time. for very large images they were able to achieve 200 times speed up.

This was achieved by using a DCT (Discrete Cosine Transform) algorithm in columns, then running through the rows of an 8x8 block within the image. This allowed the image being compressed to be broken down into 8x8 (or smaller) blocks that could be run simultaneously.

For the Huffman Encoding and Decoding for the algorithm[4]. A very efficient lossless compression algorithm which uses a table for commonly used data. The JPEG compression algorithm does its gathering by breaking down the image into 1K (kilobyte) blocks and counting all 256 bits in each block simultaneously. They then build the tree using one thread, this means the algorithm is the same as it would be on a CPU. They used the CPU algorithm here since there are only 256 bits in total and thus too small too consider creating a parallel algorithm for this step. The third step in Huffman encoding is the creation of the encoding table. The traversal of a tree could cause many issues with Shared and block memory within a GPU algorithm. The students from Toronto decided to just run this step on the CPU as to decrease risk of slowing the algorithm down.The last step for the Toronto teams algorithm is to encode the image in parallel by encoding each block and placing it in the correct sequence in the file. Parallel decoding is done in a similar fashion: The file is broken into blocks of the correct size and decoded, then placed back in the correct order.

Another group from the University of Stanford also wrote a GPU based JPEG compression algorithm with a slightly different algorithm[9]. They wrote that since the LZ77 algorithm they used does not use any explicit dictionary, the image can be broken up into blocks at arbitrary positions. Their algorithm used a principal of breaking up the image into blocks, and then running threads within each block to create their Huffman table. This allowed them to test if the number of threads within each block would cause a significant speed increase. Using 4 threads lead to a 2.5 times speed increase, and 8 threads up to around 3.6 times increase.

NVIDIA has published their own DXT compression article run on CUDA (Compute unified Device Architecture)[2]. The algorithm creates a colour table of all colours used by the image in a parallel fashion as this can be done rapidly using CUDA. They then convert each pixel into a symbol given to the specific colour in parallel. This allows for shortening of each pixel in the image to 4 bits in length, rather than 4 bits for each value within the colour, namely red, green, blue and alpha. The final section of the algorithm involves writing threads for each row placing the columns in the correct order into a string. Then linearly place the rows into the file in the correct order. Files of this sort are usually uncompressed by the program reading the image, but decompression is done just as easily and fast in the opposite order.
The DXT compression algorithm NVIDIA created achieved these results:
GPU: 54.66 milliseconds average. CPU: 563 milliseconds average.

2.2 LZO Data Compression

The LZO (Lempel-ziv-Oberhumer) algorithm is designed to work with large input data and is based on CPU based linear algorithms and works for any type of data. A Student from Obuda University created 3 possible GPU LZO algorithms[3].

In the LZO algorithm the file that is to be compressed is first cut into file-blocks at the same size as the level 2 cache of the processor. Each block is then compressed by first calculating the hash of 4 bits from the block. The hash is saved in a hash table and reformed so each has is stored within only 1 memory address. The hash table is kept smaller or equals to the size of the Level 1 cache allowing for only cache memory to be used for speed.

The best GPU based algorithm the student from Obuda University came up with starts by creating blocks from the data at the size of the Level 2 cache times by the number of multi-processors the GPU has. They then copy all those blocks memory from the RAM of the PC to the GPU memory. They then have each multi-processor run many threads that read in 4 values to create and store a hash value while another is working on the next 4 numbers and so on.

The normal CPU based LZO algorithm has a speed of 680 MB/s, while the GPU based algorithm has a speed of around 900 MB/s. The approximate increase of 30% for decompression as well.

2.3 LZSS Data Compression

LZSS (Lempel-ziv-Storer-Szymanski) is a lossless data compression algorithm for any sort of data similar to LZO. 2 Students from Delaware University wrote a article about a GPU based LZSS algorithm[7].
The LZSS algorithm does not use data in the actual file, but rather works from the binary representation. This allows it to see any type of file as if it were of the same type, allot like the LZO algorithm. This can be done since everything on a PC can be represented as a line of binary numbers.
LSZZ starts at the beginning of a stream and looks ahead to see if the next value is the same, if it is it looks even further until it reaches a value that is different, this is known as “Run Length Encoding”. It then outputs the value and how many of that that value there are in a row. It then restarts this algorithm from the next value that was different, until fully compressed.

The 2 students had come up with 2 possible algorithms:
The first was very similar to the PThread (CPU thread) algorithm. It offloads work to each thread within each block by passing a small piece of the file to work on. The LZSS algorithm has no dependencies between characters if the substring are not in the range of the same buffer. This allows the algorithm to split the work into several blocks by using the buffer length. Once distributed along the blocks. The run length encoding algorithm is run on the chunks in the blocks until finished. Each block then sends its data back to the PC for it to put back together.
The second method exploits the SIMD nature of the algorithm. The work that is spread along each blocks threads is the same phase of the algorithm. The 2 Delaware students noticed when running their first algorithm that even more of

the data is actually independent within each block. The algorithm can thus have many threads within each block work on an even smaller set of data derived from the blocks data.

They compared both their LZSS algorithms to BZIP2 and the normal LZSS algorithms:
They determined that their second version of the CULZSS (cuda ZLSS) had an average of 18 times speed up to the normal LZSS algorithm and approximately 6 times speed up compared to the BZIP2 algorithm. The algorithm does not achieve such great results when working on already highly compressed data. They found that their first algorithm was the fastest for already compressed data. Achieving 10 times faster speed than the BZIP2 algorithm and around 5 times speed up over the second CULZSS algorithm.

3. DISCUSSION

From the results of the test on algorithms mentioned above, we can determine that if the compression algorithm can be parallelized we typically gain huge time decrease by using the GPU rather than normal PThreads.
The average increase is around 20 times faster. That is a huge amount. Sadly that does all depend on the data being compressed. If the algorithm is specific for a data tpe like JPEG files, we will have huge speed increase like 20 times. But when it comes to algorithms designed to work on any type of data such as LZO, we sit around only 2 - 10 times faster.
This is also affected by the amount of parallelism the algorithm can achieve. The LZO algorithm has a lot of determinant data. Thus we can only parallelize to a certain point before having to sit with a linear step, this is shown in Amdhal's Law[5].

This also means that there are many compression algorithms out there that cannot be parallelized. And there are many algorithms that could be made parallel but not on a GPU due to their immense memory requirements. Some algorithms also have problems in how the memory is accessed, such as the DXT problem mentioned above. Another issue that should be considered is the amount to time spent passing data to and from the PC's RAM and the GPU's RAM.

This does support our theory that data could be compressed fast enough for streaming data compression. Thus the SKA issue could most likely be fixed through GPU based compression.

4. GLOSSARY

EOF	End of File
PThread	CPU threads
CPU	Central Processing unit
GPU	Graphical Processing unit
SKA	Square Kilometer Array (Radio telescope system in South Africa)
TB	Terabyte (1000 Gigabytes)
RAR	Roshal Archive (Any file compression, or multi-file compression)
JPEG	Joint Photographic Experts Group
MP3	MPEG layer 3 (Music compression)
GIF	Graphics Interchange Format (Image compression, multi-image compression)
MPEG	Moving Picture Experts Group (Video compression)
CUDA	Compute Unified Device Architecture
DCT	Discrete Cosine Transform (Video or visual compression)
LZO	Lempel-ziv-Oberhumer
LZSS	Lempel-ziv-Storer-Szymanski
SIMD	Single Instruction, Multiple Data

5. REFERENCES

[1] Yi Cao, Li Xiao, and Huawei Wang. Hardware-accelerated volume rendering based on dxt compressed datasets. In *Audio Language and Image Processing (ICALIP), 2010 International Conference on*, pages 523–527, 2010.

[2] Ignacio Castano. High quality dxt compression using cuda. In *NVIDIA ICG*, 2007.

[3] L. Erdodi. File compression with lzo algorithm using nvidia cuda architecture. In *Logistics and Industrial Informatics (LINDI), 2012 4th IEEE International Symposium on*, pages 251–254, 2012.

[4] M. Kawahara, Y.-J. Chiu, and T. Berger. High-speed software implementation of huffman coding. In *Data Compression Conference, 1998. DCC '98. Proceedings*, pages 553–, 1998.

[5] L. Kleinrock and Jau-Hsiung Huang. On parallel processing systems: Amdahl's law generalized and some results on optimal design. *Software Engineering, IEEE Transactions on*, 18(5):434–447, 1992.

[6] Wei ling Chang, Xiao chun Yun, Bin-Xing Fang, and Shu peng Wang. The block lzss compression algorithm. In *Data Compression Conference, 2009. DCC '09.*, pages 439–439, 2009.

[7] A. Ozsoy and M. Swany. Culzss: Lzss lossless data compression on cuda. In *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, pages 403–411, 2011.

[8] Pranit Patel, Jeff Wong, Manisha Tatikonda, and Jerek Marczewski. Jpeg compression algorithm using cuda. 2008.

[9] Leiwen Wu, Mark Storus, and David Cross. Cuda wuda shuda: Cuda compression project. 2009.