RESEARCH ARTICLE OPEN ACCESS

Review on Parallel Approach for Lossless File Compression on CPU using OpenMP

Malhar Ujawane [1], Ashutosh Barve [2]
Department of Computer Technology
Yeshwantrao Chavan College of Engineering
Nagpur – India

ABSTRACT
High performance computing (HPC) has proved to be a colossal computation resource for data intensive applications that require parallel processing for expeditious data analysis. In this paper, we analyze a parallel programming model using OpenMP for file compression. The intent of the paper is to explore literature on the subject and to provide a high level view of the features presented in the programming model, thus assisting lossless file compression.
Keywords :— *OpenMP, LZW, Lossless Compression, HPC*

I. INTRODUCTION

Nowadays, research done for parallelizing lossless file compression algorithms has become dormant. Even though there are advancements made in other fields using parallel computing, file compression due to its dominant serial nature has experienced a slowdown. An approach to solve this problem is discussed in this paper, where a serial lossless file compression algorithm can be implemented in a multiprocessing environment on the CPU using OpenMP.

We often happen to meet problems requiring heavy computations or data-intensive processing. The increasing volume of data generated by entities unquestionably, require high performance parallel processing models for robust and speedy data analysis. With problem size and complexity increasing, the need for parallel computing has resulted in a number of programming models proposed for high performance computing. Several parallel and distributed programming models and frameworks have been developed to efficiently handle such problems [1].

As the high performance computing techniques have increasingly become a necessity in mainstream computing, a number of researchers have done work on documenting various features in parallel computing models [1]. Although there are many parallel programming models such as OpenMP, CUDA, MapReduce and MPI, etc. Exist, OpenMP and CUDA are amongst the most popular ones as they utilize multi-threading and are also relatively easy to integrate into existing applications [1].

In this paper, we will discuss the OpenMP approach to lossless file compression on CPU. This approach is based on LZW algorithm for file compression.

II. DATA COMPRESSION

Data Compression is a technique of representing information in a compact form. The creation of these compact representations is by identifying and using structures that exist in data. Data can be characters in a text file, numbers that are samples of speech or images. Data compression attempts to identify redundancy and use it effectively for compression. Data compression consists of two algorithms. First is compression algorithm that takes input X and generates a representation Xc (compressed data) that requires fewer bits, and Second is a reconstruction algorithm that operates on compressed representation Xc to generate reconstruction Y (reconstructed data). Based on requirements of reconstruction, data compression schemes can be divided into two broad categories:

A. Lossless Compression Schemes

In this scheme, Y (reconstructed information) is exactly same as X (original information). Lossless compression is generally used for applications that cannot tolerate any difference between the original and reconstructed data. Text compression is an important example of lossless compression.

B. Lossy Compression Schemes

In this scheme, Y (reconstructed information) is not exactly as X (original information), but it is almost same as X. It involves some loss of irrelevant information. Data that have been compressed using lossy techniques generally obtain much higher compression ratios than it is possible with lossless compression. Speech, Video and Image compression are the important examples of lossy compression.

Examples are shown in the figures below. For this research, we will be focusing only on the lossless data compression algorithms.

Fig 1. High Compression (Low Quality) JPEG [2]

Fig 2. Low Compression (High Quality) JPEG [2]

C. LZW Algorithm

While many lossless compression algorithms are available that are essentially derived from the Lempel Ziv family of algorithms, namely LZ77, LZ78, LZSS, LZMA LZW etc. this paper focuses on the LZW algorithm.

Lempel–Ziv–Welch (LZW) is a universal lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch. It was published by Welch in 1984 as an improved implementation of the LZ78 algorithm published by Lempel and Ziv in 1978. The algorithm is simple to implement, and has the potential for very high throughput in hardware implementations. It is the algorithm of the widely used UNIX file compression utility compress, and is used in the GIF image format [3].

The scenario described by Welch's 1984 paper encodes sequences of 8-bit data as fixed-length 12-bit codes. The codes from 0 to 255 represent 1-character sequences consisting of the corresponding 8-bit character, and the codes 256 through 4095 are created in a dictionary for sequences encountered in the data as it is encoded. At each stage in compression, input bytes are gathered into a sequence until the next character would make a sequence for which there is no code yet in the dictionary. The code for the sequence (without that character) is added to the output, and a new code (for the sequence with that character) is added to the dictionary [3]. Fig 3. below shows how LZW works[4].

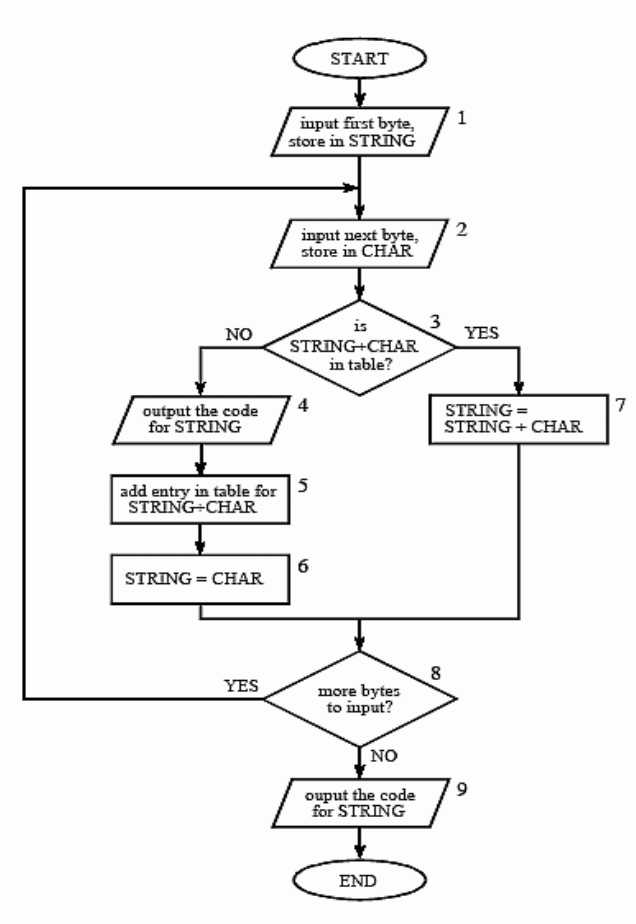

Fig 3. A flow chart of LZW compression [4]

Some reasons behind choosing LZW algorithm include:

- LZW creates a dictionary of common phrases in the data [5].
- LZW doesn't require to pass the string table to decompression code. Hence, table can be re-created as it was during compression using input stream as data [3].
- LZW can have a pointer to an infinite string but algorithms such as LZSS can only contain string pointer up to 32 bits [5].

D. OpenMP

Ref [1] highlights how we can effectively use multi-processing to accelerate common applications and increase efficiency. OpenMP is a highlighted topic because of its ease of use for CPU parallelization and ability to integrate within applications. Hence, for our research we focus on using OpenMP for file compression.

OpenMP also known as Open Multi-Processor is a shared-memory multiprocessing Application Program Inference (API) for easy development of shared memory parallel programs. It provides a set of compiler directives to create threads, synchronize the operations, and manage the shared memory on top of pthreads. The programs using OpenMP are compiled into multithreaded programs, in which threads share the same memory address space and hence the communications between threads can be very efficient. Its runtime maintains the thread pool and provides a set of libraries. It uses a block-structured approach to switch between sequential and parallel sections, which follow the fork/join model. At the entry of a parallel block, a single thread of control is split into some number of threads, and a new sequential thread is started when all the split threads have finished. Its directives allow the fine-grained control over the threads [1].

It is supported on various platforms like UNIX, LINUX, and Windows and various languages like C, C++, and FORTRAN [1]. Some of the advantages of OpenMP are:

- OpenMP is much easier to use because the compiler takes care of transforming the sequential code into parallel code according to the directives [1].
- The programmer can write multithreaded programs without serious understanding of multithreading mechanism [1].

III. PARALLEL LZW IMPLEMENTATION

The parallel implementation of the LZW algorithms uses the ability of multi-core CPUs to perform computations on each core separately. Since OpenMP allows us to execute specific parts of the program in parallel on the CPU, we use each core to process different bits of information. Steps for implementation:

- Determine the number of processors available (*n*) using OpenMP.
- Determine the size of the input file (*s*).
- Divide input file into (*s/n)* parts such that their size remains same.
- Perform encoding of bits separately (i.e compression) on each processor with OpenMP.
- The compressed intermediate output will be combined into an output file

This implementation can be explained with the following example. Consider a file size of 100 MB and an LZW algorithm with a compression rate of 5MB/sec and a quad-core CPU. In this scenario, the variables will have the following values:

Number of Processors, $n = 4$

Size of Input File, $s = 100$MB

Now the algorithm will divide the input file into four blocks each of size $(s/n) = 25$MB. Once the algorithm starts compressing the blocks at the rate of 5MB/sec, the blocks will be compressed in 5 seconds. As compared to a serial approach where the compression would require 20 seconds, this parallel implementation gives 4x boost to the serial LZW algorithm.

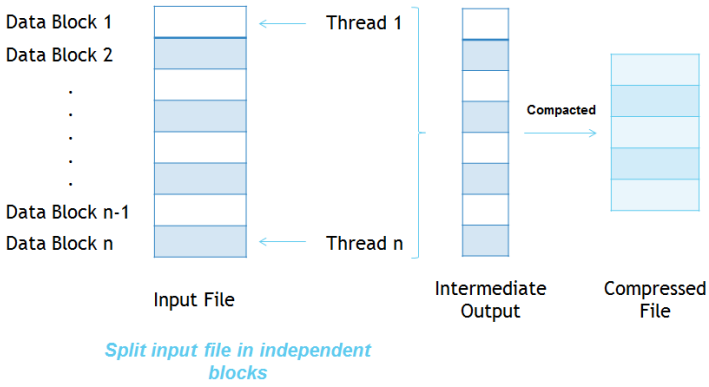

Fig 4. High-Level Overview Of This Approach

IV. CONCLUSION

After comparing with the serial implementation with the parallel OpenMP, compression time reduced significantly in a parallel computing environment about 4x-10x. Since the LZW algorithm is lossless in nature, it holds a tremendous potential for future work and integrations in other software.

The future work could be aimed at extending this approach to a GPGPU computing environment using CUDA.

REFERENCES

[1] Zahid Ansari, Asif Afzal, Moomin Muhiuddeen, Sudarshan Nayak, *Literature Survey for the Comparative Study of Various High Performance Computing Techniques.*

[2] "Lossy Compression", Wikipedia webpage [Online] *https://en.wikipedia.org/wiki/Lossy_compression*

[3] "Lempel Zev Welch Algorithm", Wikipedia webpage [Online] *https://en.wikipedia.org/wiki/Lempel-Ziv-Welch*

[4] Steven W. Smith, "Chapter 27: Data Compression", *The Scientist and Engineer's Guide to Digital Signal Processing*

[5] Yair Wiseman., *The Relative Efficiency Of Data Compression*, LZW and LZSS