Parallel Lempel-Ziv-Welch (PLZW) Technique for Data Compression

Manas Kumar Mishra[#], Tapas Kumar Mishra[*], Alok Kumar Pani[#]

[#]*Department of Computer Science & Engg, Ghanashyam Hemalata Institute of Technology & Management, Puri - 752002, India*

[*]*Department of Computer Science & Engg, Utkal University, Vanivihar, Bhubaneswar - 751004, India.*

Abstract- **Data Compression is one of the most fundamental problems in computer science and information technology. Many sequential algorithms are suggested for the problem. The most well known sequential algorithm is Lempel-Ziv-Welch (LZW) compression technique. The limitation of sequential algorithm is that ith block can be coded only after the (i-1)th block has completed. This limitation can be overcome by parallelizing the LZW coding technique. Attempt has also been made to parallelize the LZ technique [10]. But here is a new idea for parallelizing the LZW compression technique. It uses a common memory to store the encoded string parallelly in a two dimensional array and stores -1 at the end in each row which works as the marker. Similarly each row is decoded parallelly by different processors. It is suitably implemented in SMP cluster using MPI library function. The sequential algorithm takes θ(n) where n is the size of text. But the parallel algorithm takes θ (n/p) where p is the number of processor.**

Keywords- **Symmetric multi processor (SMP), message passing interface (MPI), Lempel-Ziv-Welch (LZW)**

I. INTRODUCTION

When we speak of a compression technique or compression algorithm, we are actually referring to two algorithms. There is the compression algorithm that takes an input X and generates a representation Xc that require fewer bits and there is reconstruction algorithm that operates on the compressed representation Xc to generate the reconstruction y. Based on the requirements of reconstruction, data compression schemes can be divided into two broad classes: lossless compression schemes, in which Y is identical to X, and lossy compression schemes, which generally provides much higher compression than lossless compression but allow Y to be different from X. Some of the lossless data compression techniques are Huffman Coding, Arithmetic Coding and Dictionary Techniques [12]. The drawback of Huffman coding and Arithmetic coding needs two scan, one scan to find the probability and the other to code the text. The dictionary coding may be static or adaptive. Static dictionary coding need the dictionary to be sent along with the compressed data to the receiver which is an extra headache. Some of the adaptive dictionary data compression technique are LZ77 [15], LZ78 [16] and LZW [14]. As mentioned in the abstract, parallel LZ technique[10] uses dummy bits to make the size of different encoded strings (made by each processor) of equal lengths so that decoding can be done parallely by dividing the size of encoded string with number of processors. But in this paper we use -1 at the end of each encoded string (made by each processor) which works as the marker and by which we reduces the size of the encoded string.

Cluster systems, which are groups of general-purpose computers interconnected by networks, have become very popular because of their cost/performance and scalability advantages over other parallel computing systems, such as centralized supercomputers. Although various types of architecture of cluster systems exit, the symmetric multiprocessor (SMP) clusters system has been in the mainstream. In this paper, we propose a coarse grain parallel algorithm for LZW compression technique. All message transmissions are carried out by MPI library. In the rest of the paper, section II and II(A) describe the sequential algorithm and the coarse grain parallel algorithm respectively. In section III, performance of the proposed algorithm is compared with the sequential one. Section IV gives the experimental results. Finally, section V concludes this paper.

II. METHODOLOGY

The sequential encoding/compression algorithm is shown below.
Input: a string/file - InputString
Output: The compressed string/file – encoded string

A. *Algorithm encodelzw*

```
1. Initialize the dictionary entry with single distinct alphabet
2. Set tempstring=NULL
3. for(i=startindex;i<=endindex;i++) repeat step 4 to 7
4. append the character InputString[i]to the tempstring
5. search the tempstring in the dictionary
6. If tempstring is found in the dictionary, set index = dictionary index at which tempstring is present.
7. If not found store the tempstring in the dictionary. Write the index value in the output encoded string. Set tempstring=NULL and Set i=i-1.
8. store -1 at the end of the output string
9. Exit
```

B. Parallel encoding algorithm

```
Input: a string/file
Output: The compressed string/file
P: Number of processor
    1. Initialize the dictionary entry with single distinct alphabet
    2. offset=(length of text)/(number of processor)
    3. startindex=rank*offset;
    4. if(rank==P-1)    endindex=textlength-1;
    5. else endindex=startindex+offset-1;
    6. for rank=0 to P-1 pardo
        a. encodelzw(startindex, endindex, rank)
    7. store the result in the array encodestring[rank]
    8. exit
```

Here all the processor works independently because each processor operates on different data set. Each processor is assigned a rank. If there are P processors, than the ranks are 0, 1, - - , P-1. A processor will compress what part of the input string that depends on its rank and the total size of the text. All Processors operate on same size of text (i.e. offset) except the last processor which may have to process more text. Each processor process a part of the whole string that starts with startindex and ends with endindex. After processing each processor write the encoded string into a single two dimensional array. But the row number at which the encoded string is written is different (that is same as the rank of the processor). Each processor writes -1 at the end which is used as the marker. This resolved the problem that occurs in [10]

C. Sequential decoding algorithm

```
Input: Encoded integers
Output: The original text
  • Algorithm decodelzw
      1. Initialize the dictionary entry with single distinct alphabet
      2. Set tempstring=NULL
      3. Scan an integer from the encoded string until -1 is scanned and repeat step 4 to 11
      4. If(dictionary entry in that index is not NULL) do step 5 to 11
      5. Write the dictionary entry in the output
      6. Append the dictionary entry to the tempstring
      7. For(k=0;tempstring[k]!=NULL; k++) repeat step 8 to 11
      8. Parttempstr[k]=tempstring[k];parttempstr[k+1]=NULL
      9. Search parttempstr in the dictionary
      10. If not found in the dictionary store parttempstr in the dictionary
      11. Store the remaining part of the tempstring in tempstring
      12. Exit
```

D. Parallel decoding algorithm:

```
Input: Compressed string
Output: The original string
P: Number of processor
    1. Initialize the dictionary entry with single distinct alphabet
    2. for rank=0 to P-1 pardo
       decodelzw (encodestring [rank], rank)
    3. exit
```

Here all the processor read the input string from different address location hence all processor can run concurrently. A processor decodes a string in the row number equal to its rank. For Example if the rank of a processor is ‘0’ than it decodes the row ‘0’ string of the two dimensional string. Each processor read the string until -1 is reached.

III. PERFORMANCE EVALUATION & EXPERIMENT RESULT

In this section, we shall concentrate on performance comparison between the coarse grain parallel algorithm and sequential algorithm. Assume that n is the size of text and p is the number of processors. The time complexity of the sequential algorithm is θ (n). In our parallel algorithm each processor operates on n/p data size requiring θ (n/p) time complexity. The communications only need θ (1) operations.

We implement the proposed algorithm using C and MPI library. Our experiment environment is 8 nodes cluster with each of 2.1 GHz Intel Core2duo with 4GB RAM running under Red Hat Linux. Nodes are interconnected with one Gigabit Ethernet switches. We tested our algorithm using 8 nodes and measured the wall clock time between the start and the end of the algorithm as the running time. The running time includes the execution time, the communication overhead and reading the input data from a file. The algorithm the different starting index and end index to the different processor. All the processors execute concurrently reading the string concurrently and writing the output concurrently. The running times of both sequential and parallel encoding algorithm are given in table no. 1. And the running time of sequential and parallel decoding algorithm are given in table no. 2.

TABLE 1
THE COMPARISON TIME FOR LEMPEL-ZIV-WELCH COMPRESSION TECHNIQUE USING SEQUENTIAL AND PARALLEL ALGORITHM FOR 8 PROCESSORS.
TIME: IN MICROSECONDS.

Text Size	**Sequential Encoding Time**	**Parallel Encoding Time**
100B	148	64
200B	235	66
300B	329	76
400B	420	127
500B	512	101
1KB	991	162
2KB	1925	232

TABLE 2
THE DECODING TIME FOR LEMPEL-ZIV-WELCH COMPRESSION TECHNIQUE USING SEQUENTIAL AND PARALLEL ALGORITHM FOR 8 PROCESSORS.
TIME: IN MICROSECONDS

Text Size	Sequential Decoding Time	Parallel Decoding Time
100B	87	57
200B	139	65
300B	186	72
400B	237	93
500B	287	129
1KB	554	151

The experiment is aimed at discovering how the text size affects the performance of the system, as a function of the number of hosts available for processing. In this experiment the text size varies from 100B to 2KB. X-axis represents size of text and Y-axis represents time. Fig1 shows the sequential vs parallel time for encoding and Fig2 shows sequential vs parallel time for decoding.

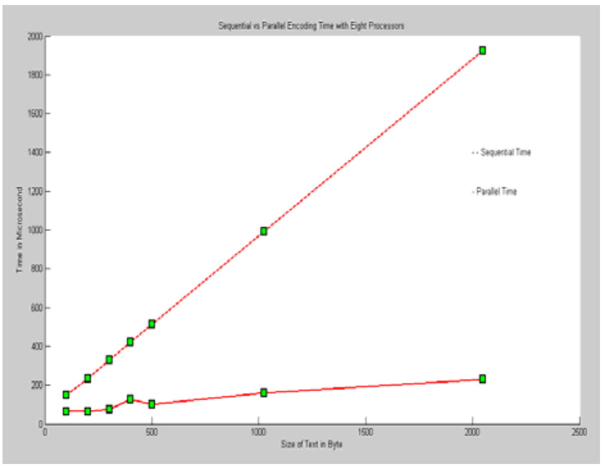

Fig. 1. Sequential vs Parallel time for encoding

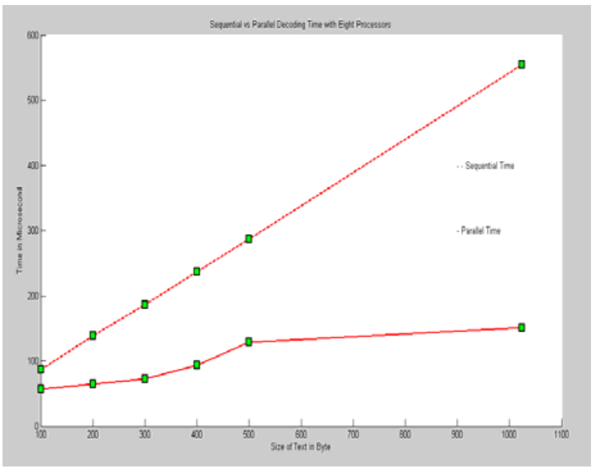

Fig. 2. Sequential vs Parallel time for decoding

IV. CONCLUSIONS

We claim that a coarse grain parallelization method on LZW compression technique can improve the existing sequential algorithms. The experimental results gave the quantitative indication to support our claim. The algorithm also showed good scalability in the sense that increasing the number of processors and text size simultaneously maintains the speedup.

ACKNOWLEDGMENT

I render my acknowledgement to Prof. (Dr.) Sudarsan Padhy for his advice, suggestions, inspiration and guidance for this work. I am also thankful to Prof. (Dr.) Ajit Nayak for his co-operation in installing MPI.

REFERENCES

[1] S. De Agostino, J.A. Storer, Parallel algorithms for optimal compression using dictionaries with the prefix property, Proceedings of Data Compression *Conference DCC–92, Snowbird, Utah IEEE Computer Society Press, 1992, pp. 52–61.*

[2] D. Belinskaya, S. De Agostino, J.A. Storer, Near optimal compression with respect to a static dictionary on a practical massively parallel architecture, Proceedings of Data Compression *Conference DCC–95, Snowbird, Utah IEEE Computer Society Press, 1995, pp. 172–181.*

[3] G. E. Blelloch and J. Greiner. Parallelism in sequential functional languages. *In Proceedings of the Symposium on Functional Programming and Computer Architecture, pages 226–237, June 1995.*

[4] G. E. Blelloch. *Programming parallel algorithms. Communications of the ACM, 39(3):85–97, Mar. 1996.*

[5] Faller. An Adaptive System for Data Compression. In Record of the *7th Asilomar Conference on Circuits, Systems, and Computers, pages 593–597. IEEE, 1973.*

[6] R.G. Gallager. Variations on a theme by Huffman. *IEEE Transactions on Information Theory, IT-24(6):668–674, November 1978.*

[7] M.E. Gonzalez Smith and J.A. Storer, Parallel algorithms for data compression, *J. ACM 32 (1985) (2), pp. 344–373.*

[8] R. M. Karp and V. Ramachandran. *Parallel algorithms for shared-memory machines. In J. van Leeuwen, editor, Handbook of Theoretical Computer Science, volume A:* Algorithms and Complexity, pages 869–941. Elsevier Science Publishers, Amsterdam, The Netherlands, 1990.

[9] Shmuel Tomi Klein and Y. Wiseman, *Parallel Huffman decoding with applications to JPEG files*, Comput. J. 46 (2003) (5), pp. 487–497.

[10] S.T. Klein, and Yair Wiseman, *"Parallel Lempel Ziv coding"*, Discrete Applied Mathematics Volume 146, Issue 2, 1 March 2005, Pages 180-191, 12th Annual Symposium on Combinatorial Pattern Matching .

[11] D.E. Knuth. Dynamic Huffman coding. Journal of Algorithms, 6:163–180, 1985.

[12] Khalid Sayood, Introduction to Data Compression, third edition, Morgan Kaufmann Publishers.

[13] J.S. Vitter. Design and analysis of dynamic Huffman codes. Journal of ACM, 34(4):825–845, October 1987.

[14] T.A. Welch. A technique for high-performance data compression. *IEEE Computer, pages 8–19, June 1984.*

[15] J. Ziv and A. Lempel. A universal algorithm for data compression. IEEE Transactions on Information Theory, IT-23(3):337–343, May 1977.

[16] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory, IT-24(5):530–536, September 1978.*