15418 Parallel Computer Architecture and Programming Project Proposal: Implementation and Comparison of Parallel LZ77 and LZ78 Algorithms

Fuyao Zhao
fuyaoz@cs.cmu.edu

Nagarathnam Muthusamy
nmuthusa@andrew.cmu.edu

1 Summary

We are planning to parallelize the LZ77 algorithm based on the its sequential algorithm described in [4] and also implement a parallel LZW algorithm which is given in [8], then we want to do comparison and evaluation on those two algorithms.

2 Background

LZ77 [9], LZW [8] (an variation from LZ78 [10]) are most common known compression algorithms. These algorithms were designed with single core implementation in mind. We are planning to find which algorithm among the LZ family is more supportive for parallelization.

[5] provides a summary of various methods used in the parallization of LZ family of compression algorithms.

2.1 Parallel LZ77

Multiple solutions existed in sequential algorithm, our solution will try to parallelize [4]. The idea is to build a suffix array, then to find the (Longest Previous Factor) for each suffix, which can be proved to the nearest smaller value in suffix array. It may need more explanation of our algorithm, but following is the steps we will take to solve it.

- Build a suffix array for the original string in parallel. [6]
- Do an All Nearest Smaller Value (ANSV) problem in parallel, and construct an array called LPF (Longest Previous Factor) [2]
- Do a variation of pointer jumper problem in parallel on the LPF. [3]
- Use parallel prefix sum algorithm to get the final result.

2.2 Parallel LZW

A binary dependency tree [7] is constructed out of the blocks of text to be compressed. In each layer of the tree the evenly indexed blocks inherit their processors from their parent block, each of oddly indexed blocks start a new sequence of blocks with processors that have not been used before.

We expect that the parallelization method described in [7] is faster than our algorithm, but our algorithm might achieve better scalability and compression ratio since we dont introduce additional constraint for the problem.

2.3 Compression Ratio

The parallel LZ77 will generate the same result as the sequential LZ77 does with unlimited compression window size.

The parallel LZW puts extra constraint on the original problems, so the result differs according to the number of processor, also there is a loss in compression ratio because the window does not include all the blocks of the prefix.

3 Challenge

3.1 Implementation

- Constructing the correct tree dependencies among the blocks of text, like starting the compression of child only after a dictionary out of its ancestors is constructed.
- Using the dictionary constructed by one processor for parent block in the compression of child by another processor with less contention.
- Choose the right version of parallel algorithm for subproblems, if different solutions existed.

3.2 Evaluation

- Evaluating the correctness of the code, our parallel LZ77 will guarantee the unique output and the parallel LZW algorithm might not yield same result for different processor numbers. (e.g. we could write an decoder for parallel LZW to check the correctness)
- Evaluating the efficiency of our implementation of two algorithms, (e.g. we could evaluate elapsed time for efficiency, but as the project going on, we might also check if we need to evaluate the memory usage)
- Evaluating the scalability,speedup and loss of compression ratio due to scaling in parallel LZW algorithm.

4 Resources

We have the sequential version of LZ77. The counterpart LZW could be also implemented. We are going to use some existed parallel algorithm library such as parallel suffix array construction and Thrust for prefix sum.

5 Goals

- Implement parallel LZ77 and LZW algorithm. We might use some simple version of parallel algorithm for particular sub-problem (e.g. for pointer jump problem, we might just use the version of $O(nlogn)$ work one instead of $O(n)$) in order to get a working implementation as soon as possible.

- To analyze the performance of the parallelized LZ77 and LZW.

If we finish ahead of time, we could do

- Optimize our implementation, replace some simple parallel algorithm for solving subproblem with more advanced one.
- Implement other versions of parallelized LZW such as [1]
- Also compare the parallel decoder associate to parallel LZ77 and LZW.
- Analyze other algorithms of LZ family like LZMA, LZSS

6 Deliverables

The report include the description of all the algorithm in details, and the experimental results. The code that implemented the algorithm for compression from LZ family suited for multi-core computers.

7 Platform

We are going to use Blacklight, and we want to use as many processors as possible.

8 Proposed Schedule

Week	What We Plan To Do	What We Actually Did
Apr 1-7	Choosing the algorithms from LZ family to be parallelized	
Apr 8-14	Implement the initial parallel version of chosen algorithms	
Apr 15-21	Optimize the Parallel version of the algorithm	
Apr 22-28	Analyze the Data collected	
Apr 29-May 5	Prepare a report on the findings	

References

[1] Belinskaya, D., and DeAgostino, S. Near optimal compression with respect to a static dictionary on a practical massively parallel architecture. In *Proceedings of the Conference on Data Compression* (Washington, DC, USA, 1995), DCC '95, IEEE Computer Society, pp. 172–.

[2] Berkman, O., Schieber, B., and Vishkin, U. Optimal doubly logarithmic parallel algorithms based on finding all nearest smaller values. *J. Algorithms 14* (May 1993), 344–370.

[3] Blelloch, G. E., and Maggs, B. M. Parallel algorithms.

[4] Crochemore, M., and Ilie, L. Computing longest previous factor in linear time and applications. *Inf. Process. Lett. 106* (April 2008), 75–80.

[5] De Agostino, S. Lempel-ziv data compression on parallel and distributed systems. In *Data Compression, Communications and Processing (CCP), 2011 First International Conference on* (june 2011), pp. 193 –202.

[6] Kärkkäinen, J., Sanders, P., and Burkhardt, S. Linear work suffix array construction. *J. ACM 53* (November 2006), 918–936.

[7] Klein, S. T., and Wiseman, Y. Parallel lempel ziv coding. *Discrete Appl. Math. 146*, 2 (Mar. 2005), 180–191.

[8] Welch, T. A technique for high-performance data compression. *Computer 17*, 6 (june 1984), 8 –19.

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

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