Parallel Processing of Grammar Compression

Masaki Matsushita and Yasushi Inoguchi
Japan Advanced Institute of Science and Technology

Introduction

- Grammar compression algorithm
  - generates CFG deriving the input text
- Re-Pair
  - representative grammar compression algorithm
  - achieves high compression ratio for text, graph and tree
  - slower than general compression algorithm in practice
    → we addressed this issue with parallel processing

Sample Application of Re-Pair

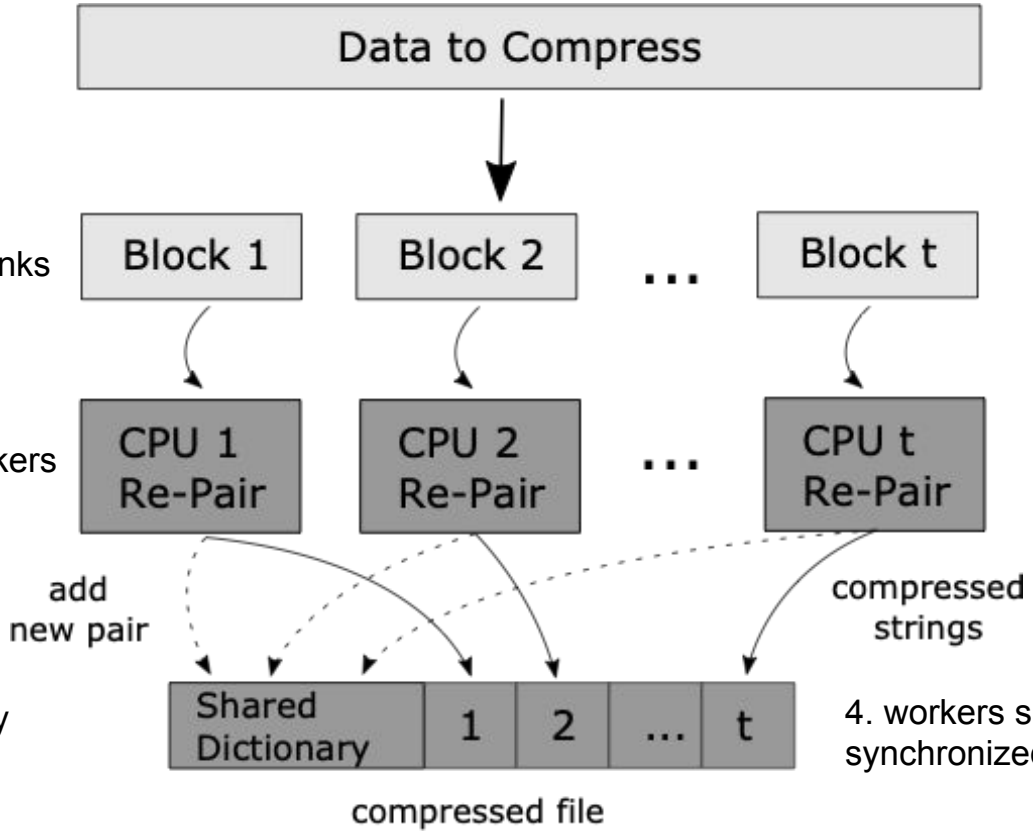
- The most frequent bi-gram is replaced by a new variable
- Output a dictionary and a compressed sequence

step	sequence	dictionary
0	abracadabra	$X_1 \rightarrow ab$
1	X_1racadX_1ra	$X_2 \rightarrow ra$
2	X_1X_2cadX_1X_2	$X_3 \rightarrow X_1X_2$
3	X_3cadX_3	

Parallel Re-Pair: a parallel variant of Re-Pair

1. breaks the input text into chunks

2. feed chunks into Re-Pair workers



3. workers share the dictionary synchronized with lock

4. workers share the dictionary synchronized with lock

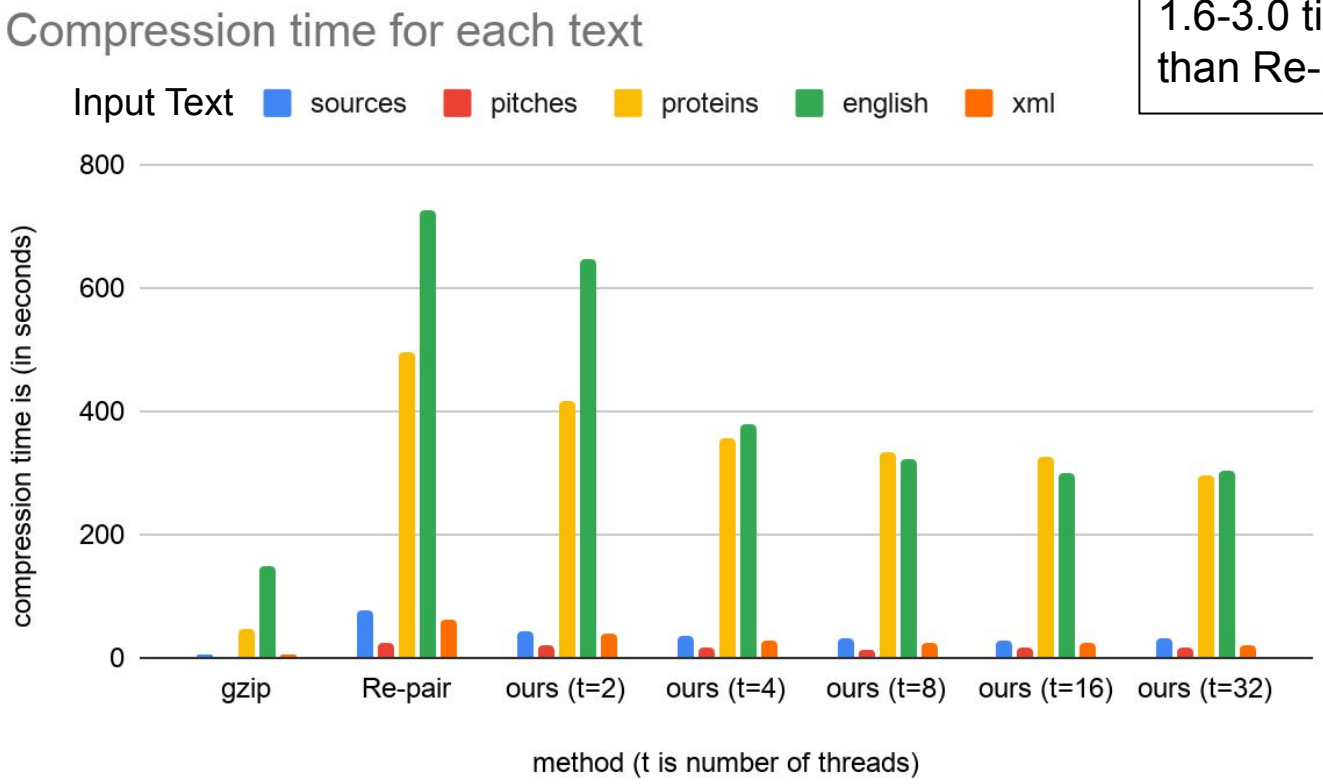
Experiments

- Implemented Parallel Re-pair with pthreads
- On dual Intel(R) Xeon(R) Gold 5220 2.20GHz with 1.5TB RAM
- We used “Pizza & Chili Corpus”

Texts	Size (MB)	Alphabet Size	Contents
sources	202	230	C/Java source code
pitches	54	133	pitch values from MIDI files
protains	1184	27	protein sequences
english	2210	239	English text
xml	283	97	XML that provides bibliographic information

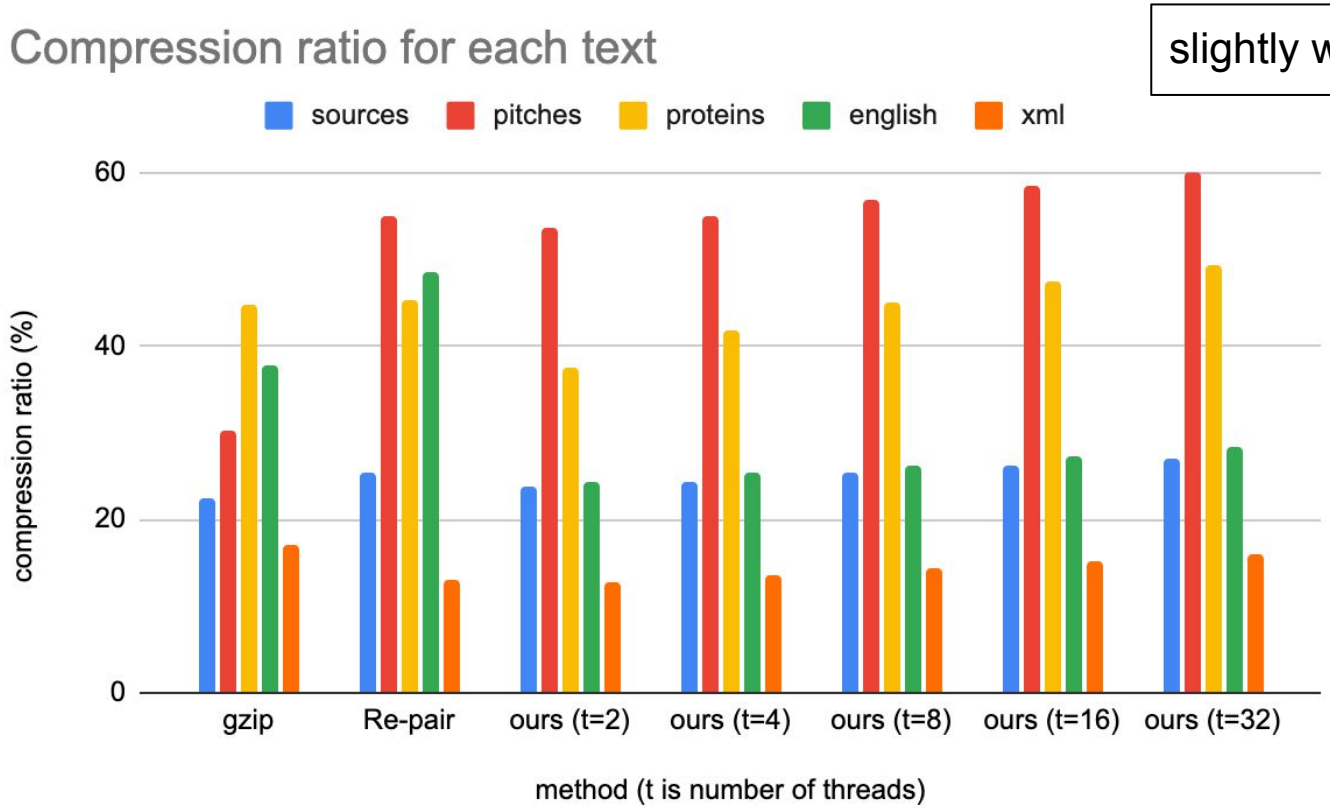
Texts used in our experiments

Experimental Results: Compression Time



1.6-3.0 times faster than Re-pair

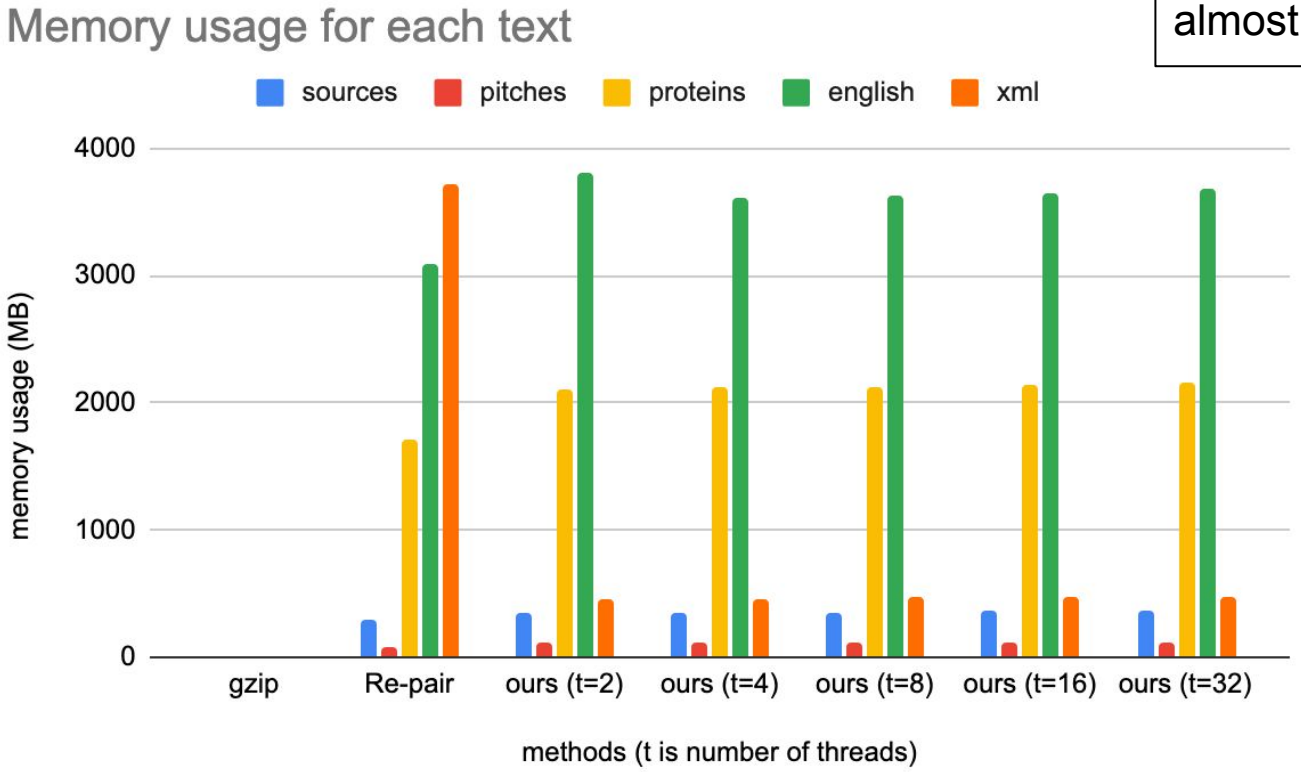
Experimental Results: Compression Ratio



slightly worse than Re-pair

Experimental Results: Memory Usage

almost same as Re-Pair



Conclusion

- We proposed a parallel variant of Re-Pair
  - simple domain-decomposition method with a shared dictionary
- Our experimental results shows
  - It is 1.6-3.0 times faster than Re-Pair
  - compression ratios are slightly worse than Re-Pair
  - memory consumption is almost same as Re-Pair
- Future work
  - improve compression time by eliminating locks