Parallel Computer Architecture and Programming

Auto-Henkan: Parallelizing Character Conversion from the English to Japanese Alphabet

Yuma Matsuoka, Joseph Wildman

Background and Objective

Background:

- No 1-to-1 mapping from English Alphabet to Japanese Hiragana
- Phonetic mapping using Romaji (English romanization of Japanese) used to type out Japanese characters on an English keyboard
- Many nuances in the language, resulting in hard algorithmic decisions and many rule exceptions
- Japanese language conversion is a very important task in the digital, internationalized era
- Not much existing literature on parallelizing this task

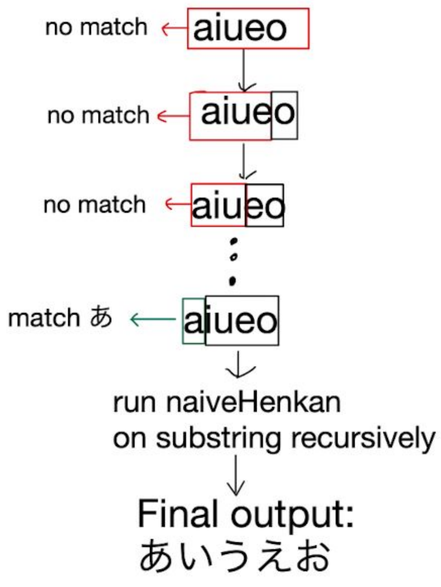
Objective:

- Develop a correct, sequential algorithm for converting english characters (Romaji) into Japanese characters (Hiragana)
- Develop a parallel algorithm and benchmark it against the sequential algorithm on the PSC BRIDGES-2 and GHC machines.
- Create an interactive demo where live user input is changed into Hiragana

Approach

Naïve Approach:

- **Sequential**, recursive approach from the start to end of a string
- Right to left scan to find a partitioned substring found in the dictionary



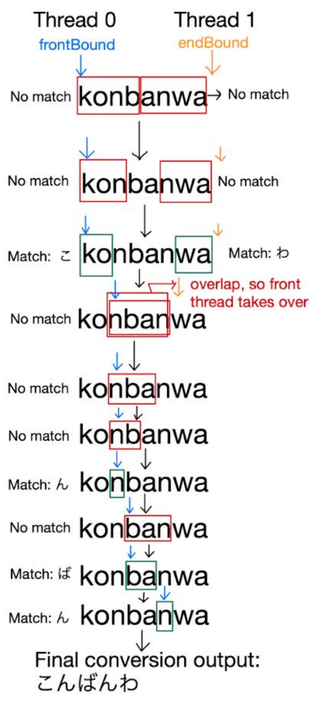


Figure 1 (Left) : Naive Henkan Approach
Figure 2 (Right): Edge Henkan Approach

Final Parallel Solution:

- Partition conversion work based on individual words in order to allow for **data-parallelism**
- Two `omp section` threads for each "word" converting from left-to-right and right-to-left simultaneously (Fig. 2)
- **Load-balance** work among threads by enforcing minimum character lengths for bins, **amortizing overhead** of spawning threads, for **two levels of parallelism** (Fig 3)
- **Granularity bound** to determine when setup of parallelism overhead is needed

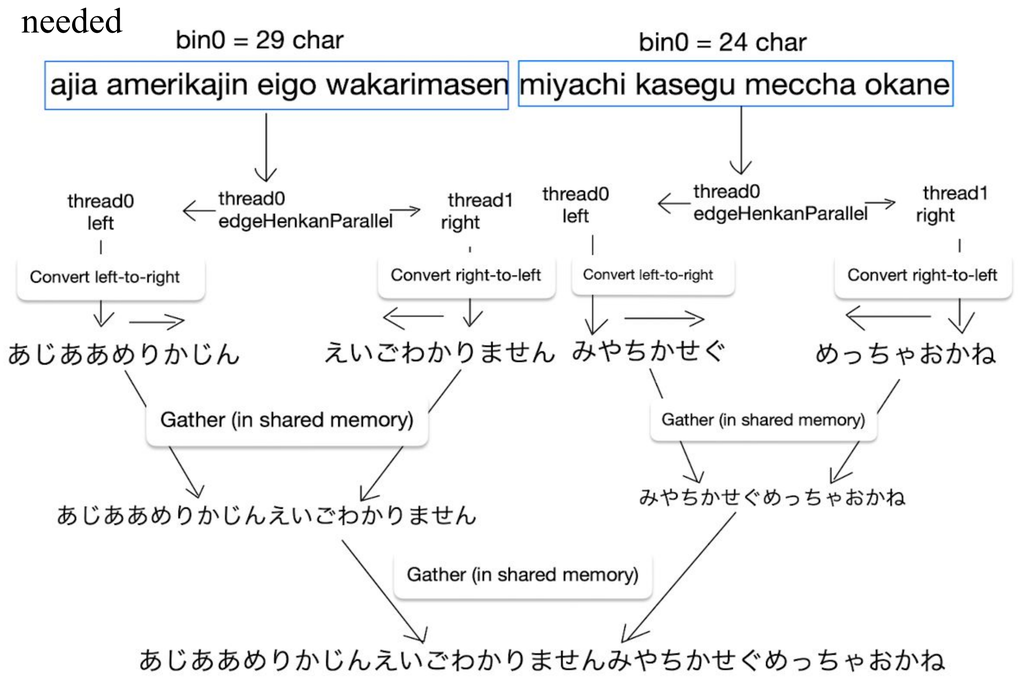


Figure 3: Load Balancing Henkan with Word and Character Parallelism

Benchmarks

Test Case	Length	Source
basic.txt	61	Japanese Characters [1]
eigo-wakarimasen.txt	454	MIYACHI - Eigo Wakarimasen [2]
newsarticle-772.txt	772	クレムリン無人機 ロシア“米指示しウクライナが攻撃”と主張 [3]
newsarticle-2500.txt	2500	クレムリン無人機 ロシア“米指示しウクライナが攻撃”と主張 [3]
longarticle-135176.txt	135176	AIリスクにどう向き合う？ [4]
really-long-article-54000.txt	540000	AIリスクにどう向き合う？ [4]

- Taken from popular media to offer a normal distribution of the characters being converted as well as a rich variety of grammatical exceptions to be handled

Results & Discussion

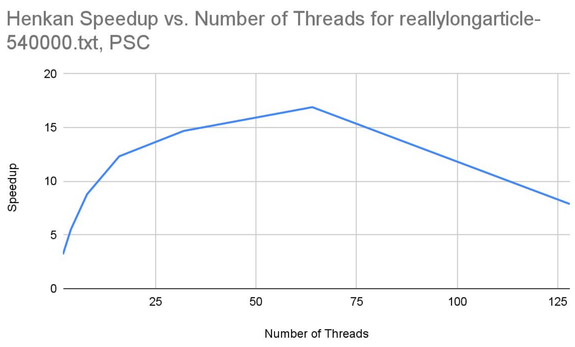


Figure 4: Speedup vs. number of threads for 540k-length input

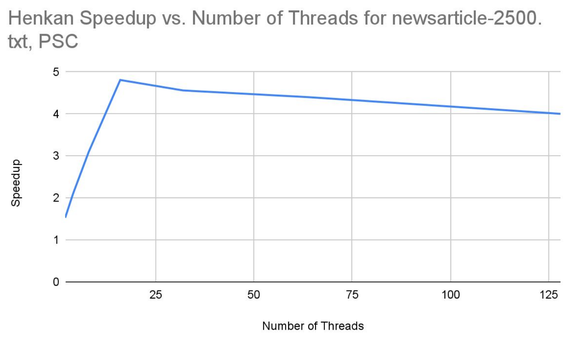


Figure 5: Speedup vs. number of threads for 2.5k-length input

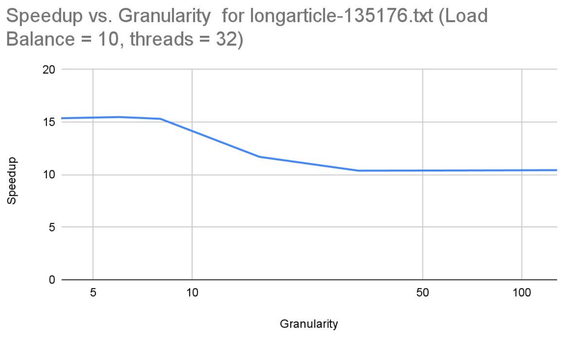


Figure 6: Speedup vs. Granularity Threshold

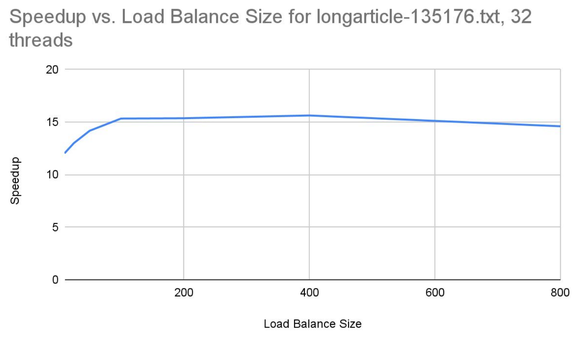


Figure 7: Speedup vs. Load Balancing Size

- 16.9x speedup on PSC machines for really large outputs (135k,540k length inputs) at 64 threads
  - More threads would make the overhead required to manage threads outweigh the benefits of parallel work
- 4.8x speedup on PSC machines for medium sized outputs (2.5k length inputs) at 16 threads
  - Less work for threads to efficiently divide and conquer
- Ideal granularity bound of size 16
- 200 character-sized task division benefits the most for balancing work among processors with enough work being done to benefit from parallelism
- Higher speedup not reached due to UTF-8 encoding of Hiragana (3 bytes per character). Spans across cache lines and a fully encoded Japanese alphabet (1-2 bytes per character) benefits from less cache miss and fetch clock cycles

Conclusion

- Project deliverables completed, over 10x speedup with algorithmic correctness, as well as a live demo
- Parallelization of character conversion is difficult due to low arithmetic intensity, and the memory-intensive nature of the program
- Even with load-balancing, overhead of spawning threads cuts into speedup gains
- Dedicated Hardware, more compact encoding of Japanese Hiragana (instead of UTF-8) could help alleviate these issues