The program works as follows:

- $S_1$: File manager locates all available files and stores them in an array.
- $S_2$: Initiates threads based on number of files and processors available.
- $S_3$: Thread reads the files and tries to find the number of syllables for each word. When all words have been processed, it updates all participants’ information.
- $S_4$: Calculates the number of syllables and creates an object of $S_4$.
- $S_5$: Stores various attributes for a word (i.e., syllable count, location, length of sentence, etc.).

The efficiency of the program was tested to confirm scalability and time-efficiency. The program should support larger transcript volumes and additional linguistic testing functions. First, eighty-eight transcripts were tested for thread processing time on strings versus threads. Referring to figure one, this would mean that $S_5$ would read the files, store the strings and send the strings to $S_4$ for processing. When sending over files, the average processing time was 0.25 seconds. When sending over strings for processing, the average completion time was 1.50 seconds. For this reason, testing continued on the program that used files for transferring, as initially explained in Figure one.

Next, duplicate files were randomly generated from the language corpus to simulate the processing of a larger set of files. The goal was to test the efficiency of the program with multiple threads. This bootstrapped dataset is a simulated representation of processing times, providing approximate processing times at each scale.

The program was run on various data sizes, as shown in table five. Table six includes the completion time for each set of file sizes and number of cores. Generally, more files can be processed in a shorter period of time as the number of threads increases. The only exceptions are the completion times for three and four threads on 1,750 files. The number of files was divided in this manner because no more than 1,800 files could be processed with the computers given memory. For three and four threads, the completion times are 19.14 and 19.54 where two threads is 18.49 seconds. This does not follow the trend from the previous completion times.

The interview parse times were evaluated to see if there was a significant difference in completion times. Each thread was able to parse a transcript (removing single-letter words and symbols) in under a second. The next possible explanation could be that since the text was being split for each thread by the number of lines, some threads may process more words than others. The counts for each word processing per thread are included below.

As shown, the threads had at maximum, a +5 word difference. Taking these results into account, the next area to investigate is the influence of memory on processing time. A delay in processing may be attributed to the fact that 1,750 files were approaching the memory processing limits of the machine. In other words, the processing times may become less efficient as the computer reaches its peak memory allocation.

The patterns for completion time and their changes can be seen on the plot in Figure two. Overall, the program shows the potential to be scalable as the number of files increases. However, more exploration is needed to verify that the changes in completion time are truly attributed to the limits of the testing environment. The results for speedup and efficiency are not as powerful as they may be but as more linguistic testing functions are included into the program, the scale of these results may improve.

## METHODS

### RESULTS

The program was run on various data sizes, as shown in the table below. Table three includes the completion time for each set of file sizes and number of cores. Generally, more files can be processed faster in a shorter period of time as the number of threads increases. The only exceptions are the completion times for three and four threads on 1,750 files. The number of files was divided in this manner because no more than 1,800 files could be processed with the computers given memory. For three and four threads, the completion times are 19.14 and 19.54 where two threads is 18.49 seconds. This does not follow the trend from the previous completion times.

The interview parse times were evaluated to see if there was a significant difference in completion times. Each thread was able to parse a transcript (removing single-letter words and symbols) in under a second. The next possible explanation could be that since the text was being split for each thread by the number of lines, some threads may process more words than others. The counts for each word processing per thread are included below.

As shown, the threads had at maximum, a +5 word difference. Taking these results into account, the next area to investigate is the influence of memory on processing time. A delay in processing may be attributed to the fact that 1,750 files were approaching the memory processing limits of the machine. In other words, the processing times may become less efficient as the computer reaches its peak memory allocation.

The patterns for completion time and their changes can be seen on the plot in Figure two. Overall, the program shows the potential to be scalable as the number of files increases. However, more exploration is needed to verify that the changes in completion time are truly attributed to the limits of the testing environment. The results for speedup and efficiency are not as powerful as they may be but as more linguistic testing functions are included into the program, the scale of these results may improve.

## REFERENCES


