N = 2	RUN: 1	RUN: 2	RUN: 3	AVE TIME(S)
THREAD COUNT: 1	1.98s	1.98s	1.95s	1.97s
THREAD COUNT: 2	0.99s	0.99s	1.00s	0.99s
THREAD COUNT: 4	0.68s	0.50s	0.68	0.62s
THREAD COUNT: 8	0.36s	0.35s	0.36s	0.36s

N = 3	RUN: 1	RUN: 2	RUN: 3	AVE TIMES(S)
THREAD COUNT: 1	2.10s	2.17s	2.11s	2.13s
THREAD COUNT: 2	1.35s	1.32s	1.33s	1.33s
THREAD COUNT: 4	0.83s	0.84s	0.95s	0.87s
THREAD COUNT: 8	0.70s	0.70s	0.69s	0.70s

N = 4	RUN: 1	RUN: 2	RUN: 3	AVE TIME:
THREAD COUNT: 1	5.08s	5.10s	5.08s	5.09s
THREAD COUNT: 2	3.91s	3.92s	3.89s	3.91s
THREAD COUNT: 4	3.43s	3.41s	3.45s	3.43s
THREAD COUNT: 8	3.36s	3.26s	3.27s	3.30s

Speedup	THREAD COUNT: 2	THREAD COUNT: 4	THREAD COUNT: 8
N = 2	0.98s	1.35s	1.61s
N = 3	0.80s	1.26s	1.43s
N = 4	1.18s	1.66s	1.79s

Analysis

General Strategy

In order to make pr4_p run in parallel, I created a new way of chunking the file that we had not done in class. The concept remained the same, with some added details. First, I broke up the file into chunks based on the file size and number of threads requested (chunks_requested = file_size / num_threads). This gave me an approximate area to put my start and stop points for each thread. However, since I did not want to start the ngram processing in the middle of a URL, I had to refine my chunks. I had to traverse the outer ends of the chunk in order to start after a \n and end at a \n character. This allowed me to evenly break up the file without separating URLs across threads. I did this processing in main for simplicity. I also found that in order to correctly run this program in parallel with significant speed gains, a local hash table per thread was required. This was simple to implement. Using local hash tables required the use of MUTEX lock/unlock when transferring the local data the global hash table. The transfer was simple, but I ran into issues when I freed the tables. I was crashing the program (after producing the correct results, but still not optimal) because of a double free() condition. I eventually found and removed this bug.

Testing my program was a simple process. I handled for many user inputs and tested those conditions to ensure an error was printed and the program exited if necessary. Matching my outputs to class provided inputs was a good sanity check as well. For edge case testing, I ran the program on smaller custom designed test files where I purposely attempted to break the program.

I did change how I parsed through the URLs looking for ngrams in this project. I made this change because I was no longer using fgets to grab each line in the file and process it. Since I was using a memory mapped file, I had to work with the file in a continuous manner. I did this with for loops, where I essentially viewed the file through a ngram sized window, analyzing each new ngram as int i incremented.

Benchmarking Results

For any ngram size, I consistently had speed improvements when utilizing more cores. The speedups from 2 to 8 cores was fairly consistent at 40% 44% 34% (2,3,4 grams respectively), with an average of a 40% speedup. Based on these observations, the speedups decrease as n increases. Based on my observations, factors that can effect my programs performance are ngram size, number of cores (8 seemingly optimal for large files), and file size.

Impact and Personal Reflection

Based on my results, parallel computing presents significant potential to benefit deep learning models. As deep learning models require massive mathematical calculations to 'learn', parallel processing could break up the work load between threads and allow for faster computation times. Perhaps the biggest limitation of pthreads as opposed to other parallel libraries is portability. Pthreads only runs on linux machines and therefore provides little flexibility when working with different operating systems. The concept I struggled with the most was creating a solution that allowed me to correctly chunk up the file at ‘\n’ characters. Since I used fgets on part 1, this was a new problem I had to solve. The most valuable thing I learned from this project was general knowledge of creating a parallel program. I understand the benefits of parallel computing’s speedups and the downfalls of added complexity. I think I will be able to make an informed decision on whether or not to develop a program in parallel.