Mikel Matticoli
CS3013 - Operating Systems - A2018
Project 4: Timing Test Results

Test System

These tests were run on the linux shell server located at ccc.wpi.edu. The system's CPU configuration is dual-core, Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz. At the time of testing, approximately 500mb of RAM were available for processing. No page faults were reported by the operating system, and that metric has thus been omitted for the purposes of this report.

Test Script and Configurations

The tests were run sequentially by a simple bash script, included in the files for the project submission (filename: timingTests). The following test files were used, with the corresponding file sizes listed below in bytes.

makefile	proj4.c	beeMovieScript.txt	hamlet.txt	big.txt
84	9509	59875	191734	6488666

The output of the test script can be found in timingTestResults.txt, and the data tables/graphs are included in the subsequent pages.

Single-Threaded Timing Tests

The first and most obvious thing to note from the graphs is that the wall clock time of the process increases with input file size. This makes sense for a linear search through the file. It is also worth noting that all of the wall clock times are substantially larger with a 1-byte chunk size than any of the other chunk sizes. For 1K, 4K, and 8K chunk sizes the performance differences appear to be fairly negligible across different chunk sizes. **In general however, it appears that larger chunk sizes result in faster execution times, particularly for larger files.** This would explain why using mmap() results in the fastest execution for *big.txt* - this is functionally equivalent to reading in the file with a chunk size equal to the size of the input file. It is also possible that the low-level implementation of mmap() is faster than read() for reading files into memory.

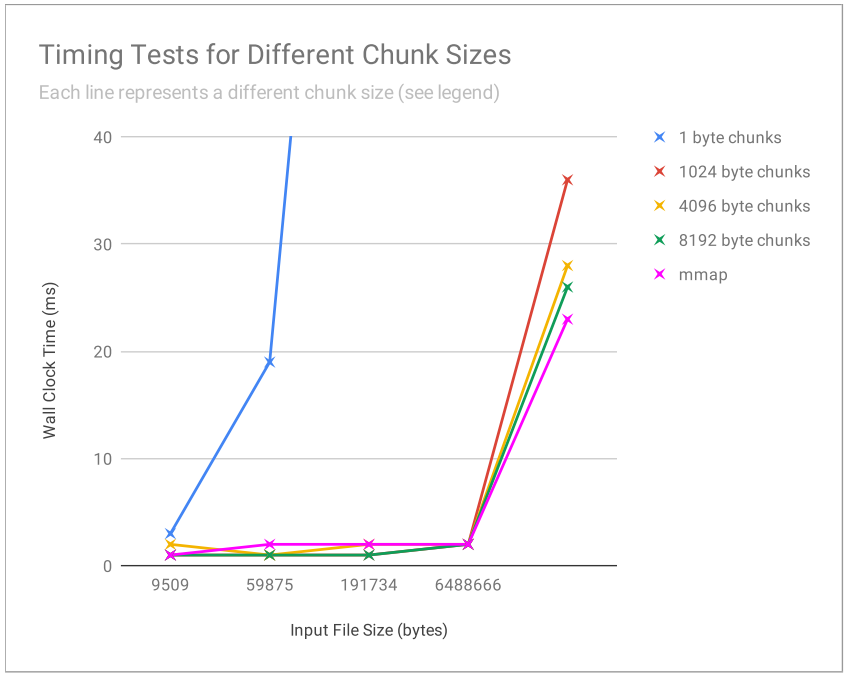
Multi-Threaded Timing Tests

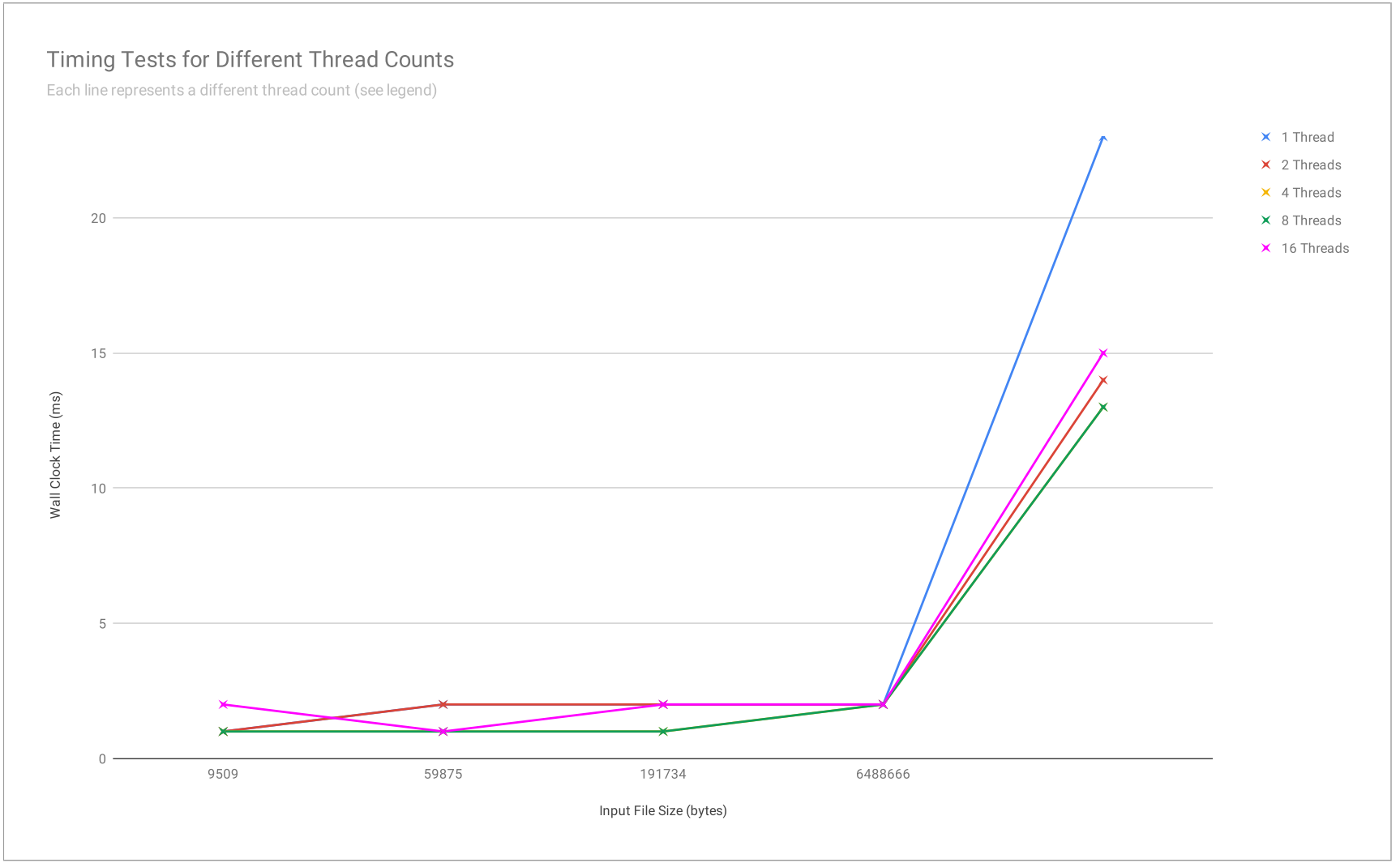
For the smaller files, the thread count again appeared to be negligible as far as total execution time. The variations in timings are likely due mostly to preemptions and other concurrent processes being run by other users on shared hardware resources. **For *big.txt*, the only noteworthy difference was that 1 thread took longer to process than multiple threads, although the number of threads past 2 appears to be irrelevant.** This makes sense, seeing as the underlying system is dual core and no more than 2 of the threads can be running concurrently anyway.

Conclusions Summarized

- Larger chunk sizes and using mmap() will result in faster execution times
- Parallelization does improve performance for large files, within the constraints of the hardware

File =>	makefile	proj4.c	beeMovieScript.tx	hamlet.txt	big.txt
File Size (bytes) =>	**84**	**9509**	**59875**	**191734**	**6488666**
1 byte chunks	3	19	118	362	12196
1024 byte chunks	1	1	1	2	36
4096 byte chunks	2	1	2	2	28
8192 byte chunks	1	1	1	2	26
mmap	1	2	2	2	23
1 Thread	1	2	2	2	23
2 Threads	1	2	2	2	14
4 Threads	1	1	1	2	13
8 Threads	1	1	1	2	13
16 Threads	2	1	2	2	15




Timing Tests for Different Thread Counts
Each line represents a different thread count (see legend)
1 Thread
2 Threads
4 Threads
8 Threads
16 Threads
20
15
10
5
0
Wall Clock Time (ms)
9509
59875
191734
6488666
Input File Size (bytes)