1. Sequential Program:

The solution to this classical “needle in the haystack” problem has been developed with Boyer Moore search available in the Boost C++ libraries. The text in the webpage has been downloaded into a text file (haystack.txt) and converted into a string using string constructor with istreambuf_iterator(). Then I erased all the whitespace in the file.(This is done to match the parameter “MarkTwain” with “Mark Twain” in the text. Also, you can search multiline patterns). Using the boyer_moore_search() that returns an iterator to the start of the pattern in the text, we loop till it reached the end of the text. Since we use the text with all whitespace removed, calculating the line number of the matched pattern requires some workaround. We then store the line number of the matched pattern in a integer array(line_nos). We measure the time interval between the start of text file reading and search loop and calculate the elapsed time.

Execution time:

single word (“England”): 0.506 sec.
Multiple word (“MarkTwain”): 0.308 sec.
Multiline(“couldn’tstandit”): 0.318 sec.

2. Parallel program (Pthreads):

The parallel implementation also uses Boost libraries for the pattern matching. Here, we convert the text file into string as above and split them into equal sized chunks (using findChunks() function) having lines equaling (NUM_LINES/NUM_THREADS). The Pthread library is used to create joinable threads. Thread argument takes the form of a structure(search_params) which is used to store all the shared variables for the threads. The threads are synchronised using mutex variables (mutexsum) which is used when writing the line number into the array(line_nos). The thread start routine (searchString()) uses the same logic as sequential part except that line numbers are calculated relative to the chunk first and then converted relative to the original text. In main() the threads are joined and the timer is called off after calculating the number of occurrences of the pattern. The results are printed along with the time taken.

Execution times:

Threads	Time (sec)	% improvement
1	0.326	- 24.42%
2	0.265	- 1.14%
4	0.262	0% (best)
6	0.271	3.43%
8	0.275	4.96%