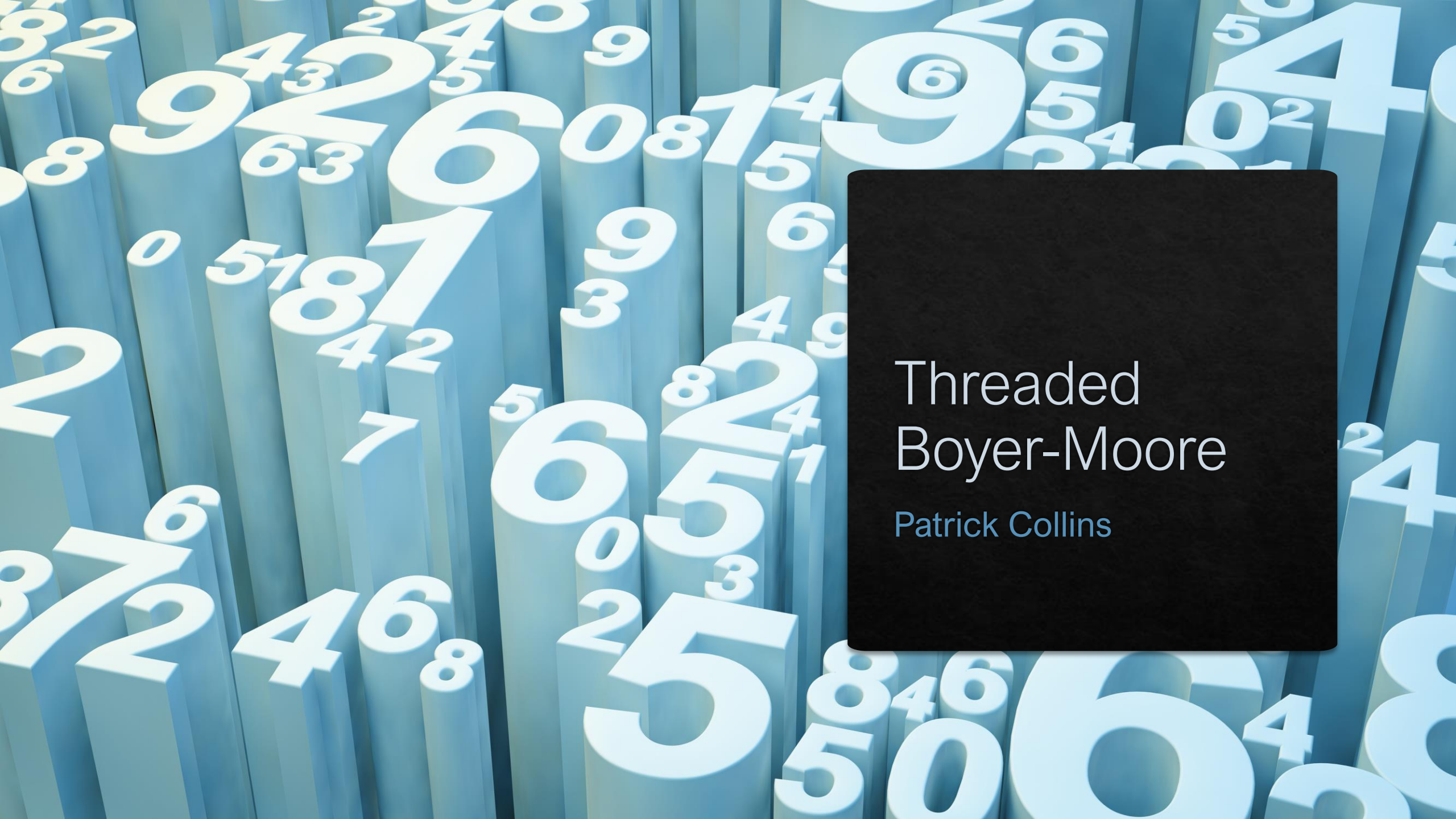


Threaded Boyer-Moore

Patrick Collins

Problem that I'll be solving

- Purpose of application: Parallelising Boyer-Moore Horspool search algorithm.
- Will achieve through parallel programming, doing a threaded task farm system.
- Once the task farm can find desired text whilst modifying thread amount, I will compare them to find out if running more threads helps speed up the Boyer-Moore algorithm.

Parallel Sections

- Farm::run() – All thread tasks run this at the same time to be told to search their section. The text amount each thread has to search through is divided to be the same amount.
- Stringsearch::run() – this is where the actual searching takes place. Each task and thread goes here to search their part of the text.

How my application makes use of threads.

- Splitting up the text to search through, and searching their section of text all at the same time. Uses substr() to achieve this.
- However, in a rare chance, it may cut off the pattern during this split, therefore not finding all matches.
- Implemented to hopefully speed up algorithm search.

Thread Safety

- Mutex – in stringsearch::run() resetting results found from previous thread before continuing.
- Unique mutex – When adding tasks, locking before running each task, adding total matches for pattern from each thread.
- Channel – for adding returned time from each thread. In Farm::run().
- Checking if collection is empty – so queue.front() isn’t called when empty.

Data Structures

- For storing matches I will be using a list. Inserting at the back using push_back() as this will be constant time O(1).
- The skipping will be done using an array as accessing the contents will be constant time O(1).

Expected Results

Goals

Boyer Moore should be much faster with more threads running.

Box plots

Boyer Moore with 8 threads should be relatively close together, as the program should run at relatively same speed.

Running with 1 thread the times may vary a lot. No consistency.

Time complexity

Boyer-Moore-Horspool: Best case O(N/M), worst-case of O(NM).

Only parallelising the algorithm, not modifying it. Time Complexity should stay the same.

Performance Evaluation

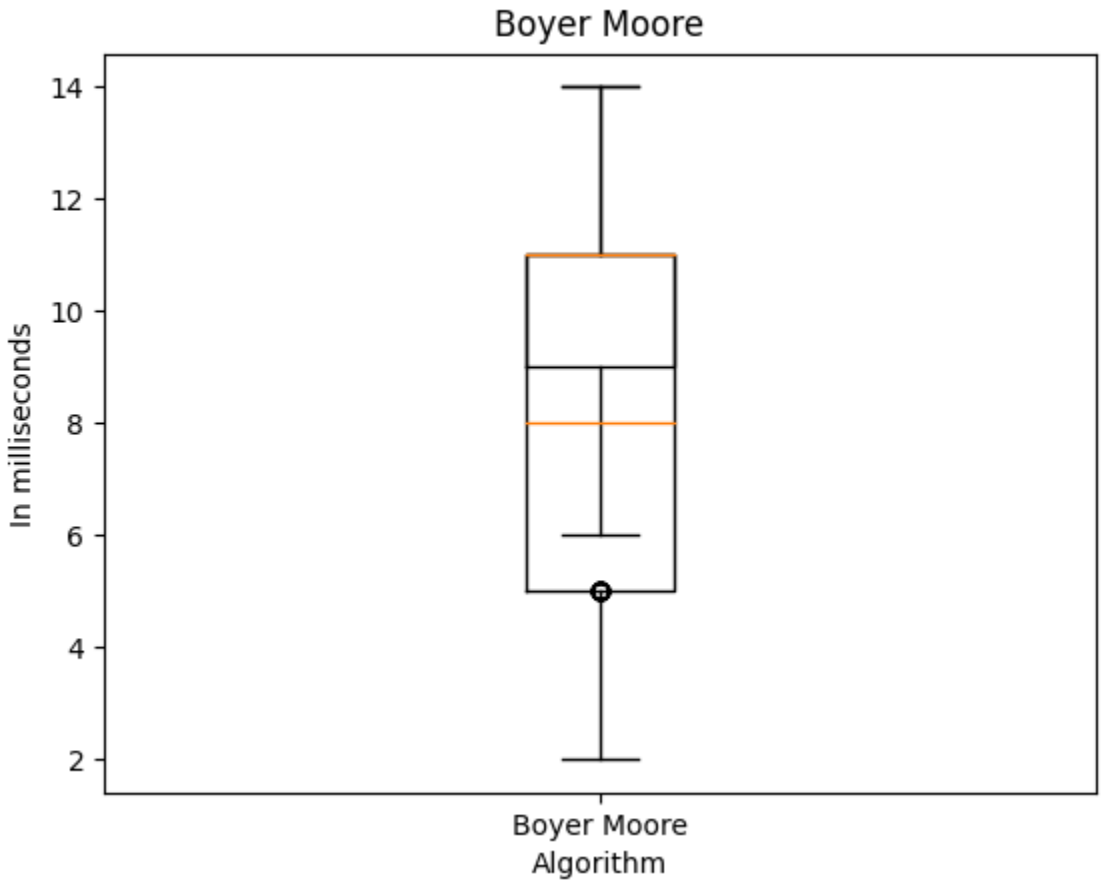
- 1 thread (177377 range of text), 1 task.
- 8 threads(22172 range of text for each thread), 8 tasks.
- Timing how long each thread takes in microseconds and adding+dividing to get total time in milliseconds.
- CPU performance evaluation to be done on: Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz
- Base speed: 1.80 GHz
- Cores: 4
- Logical processors: 8
- In the application, the user will not be able to start more threads than their CPU's Logical processors.

Bigger text file

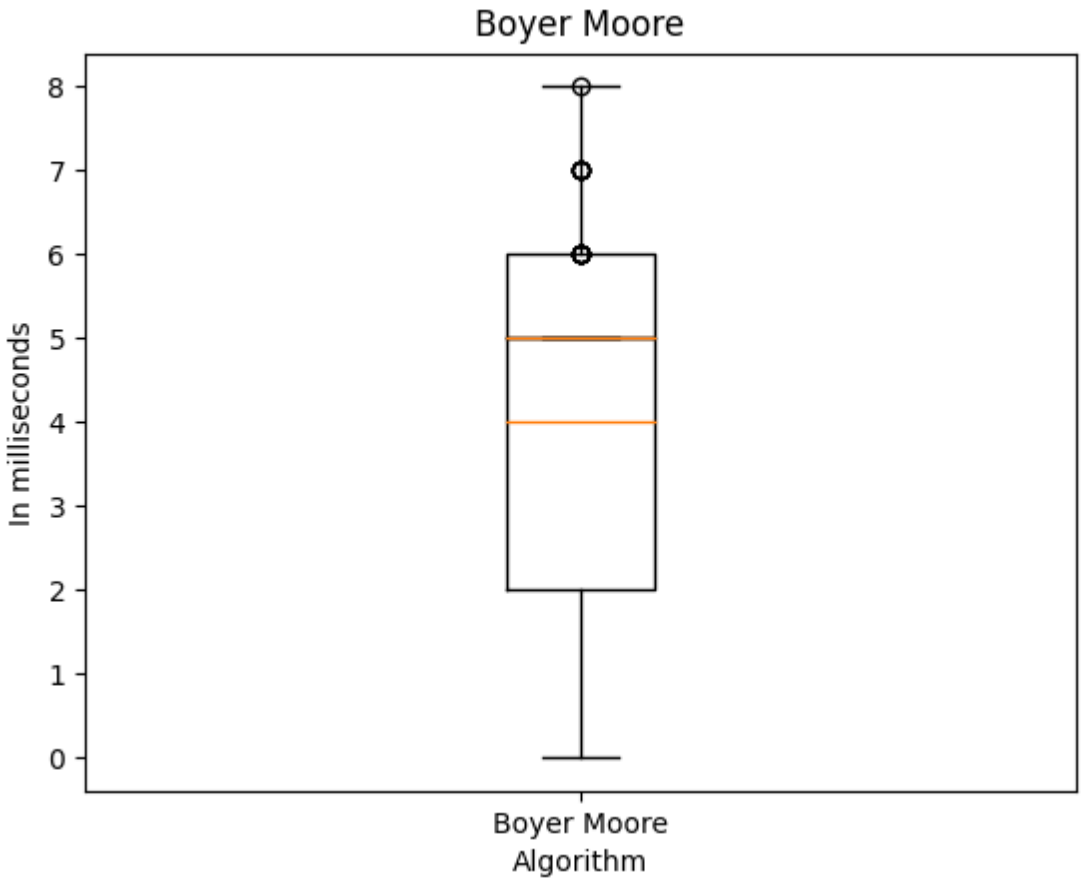
```
void load_jute_book(string& str) {
    // Read the whole file into str.
    load_file("jute-book.txt", str);

    // Extract only the main text of the book, removing the Project Gutenberg
    // header/footer and indices.
    str = str.substr(0x4d7);
}
```

Pattern=the (1,000 iterations)

1 Thread

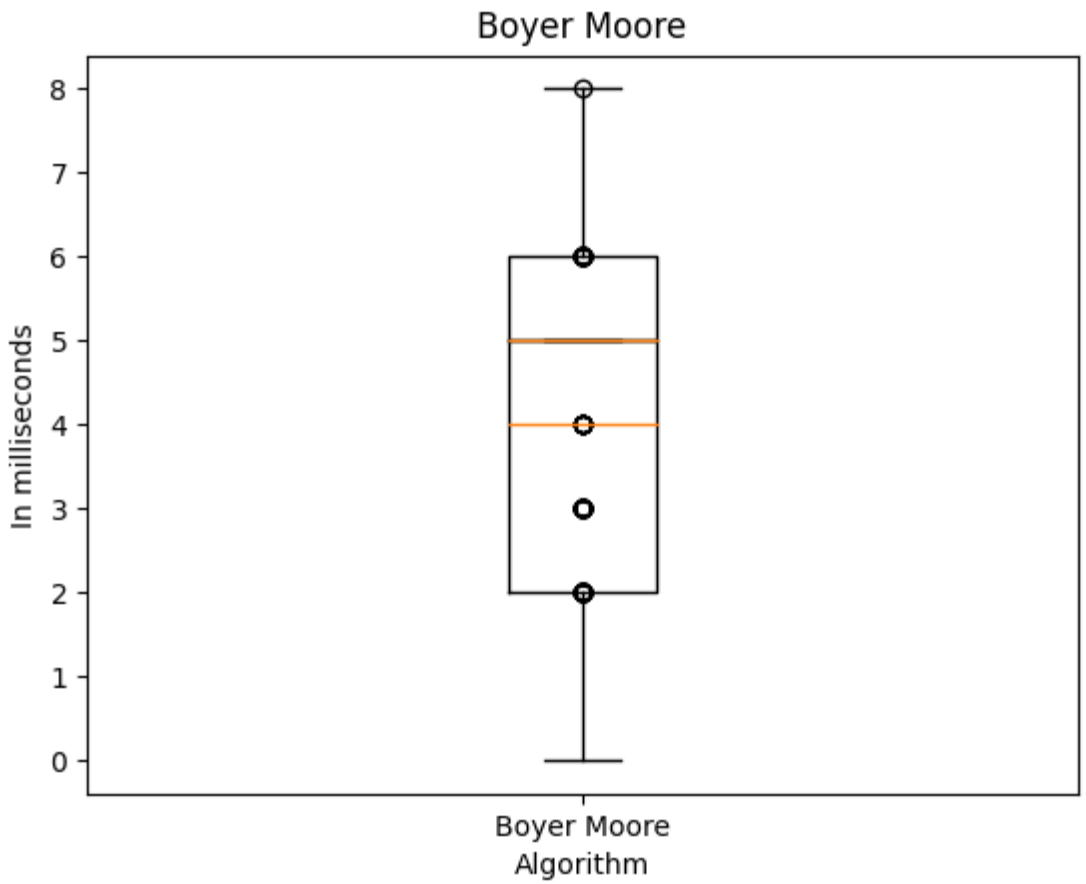


8 threads

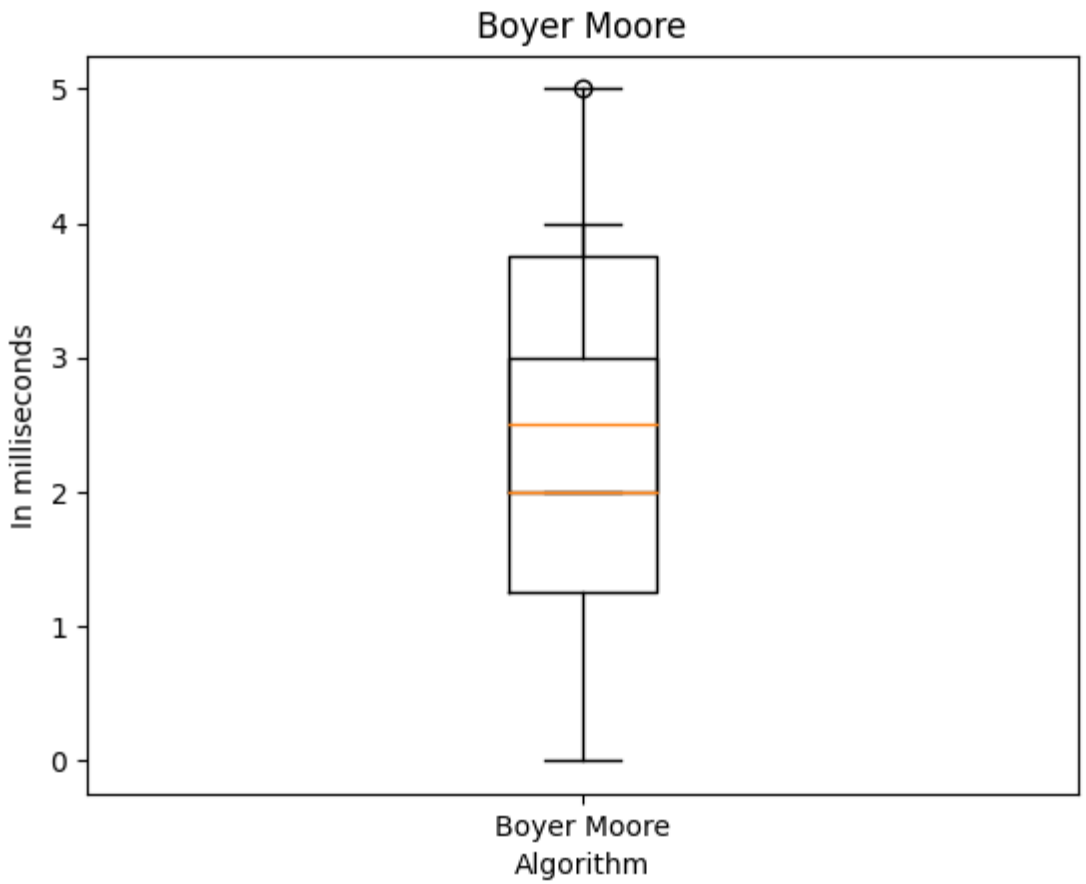


Pattern=Dundee (1,000 iterations)

1 Thread

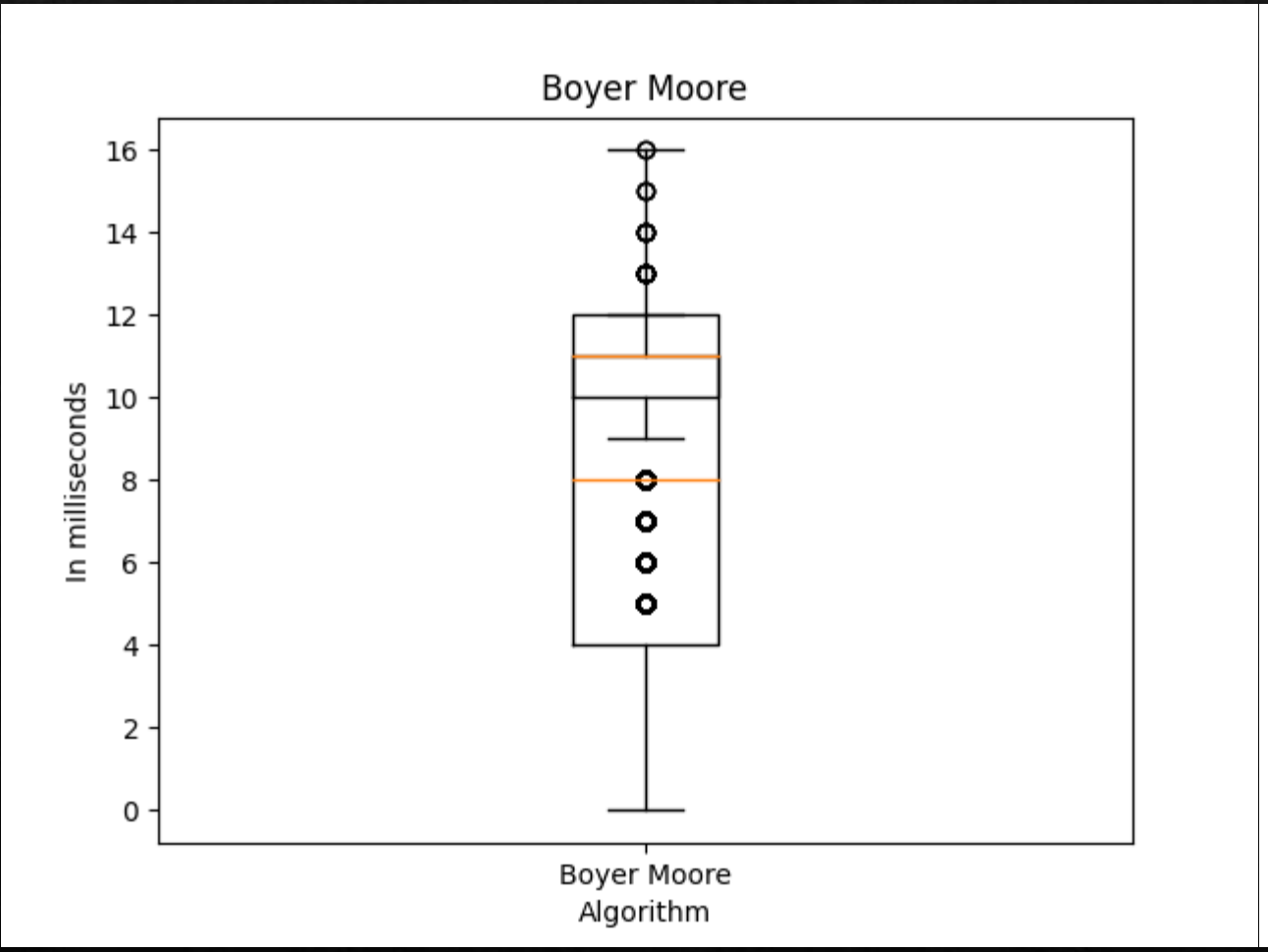


8 threads

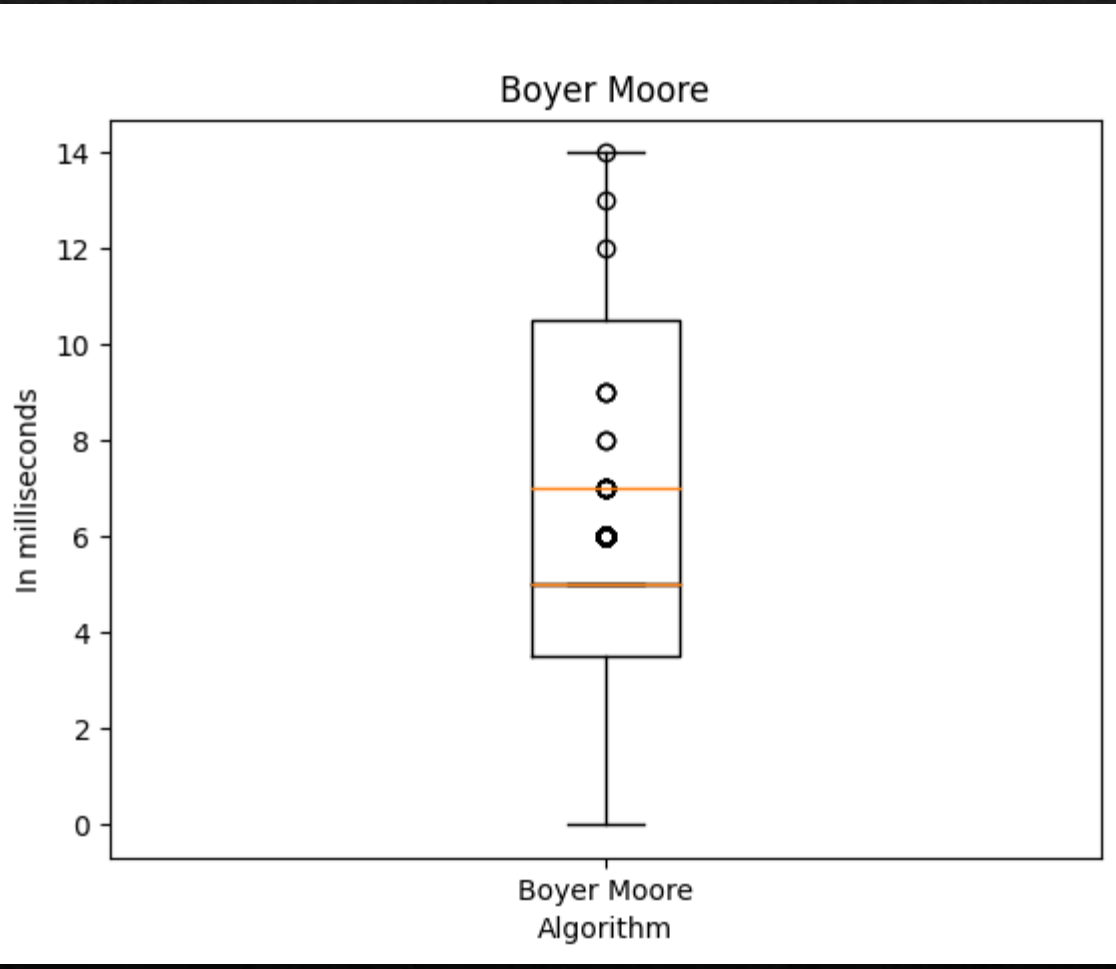


Pattern=the (10,000 iterations)

1 Thread

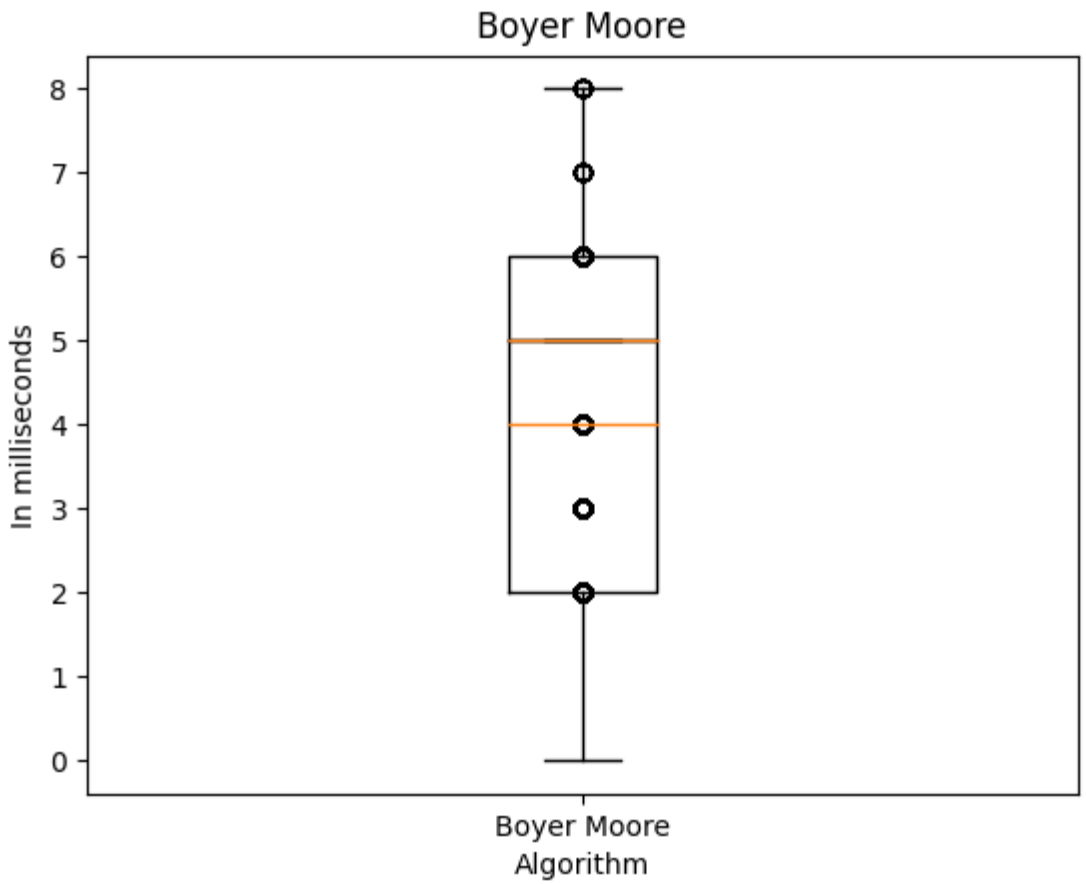


8 threads

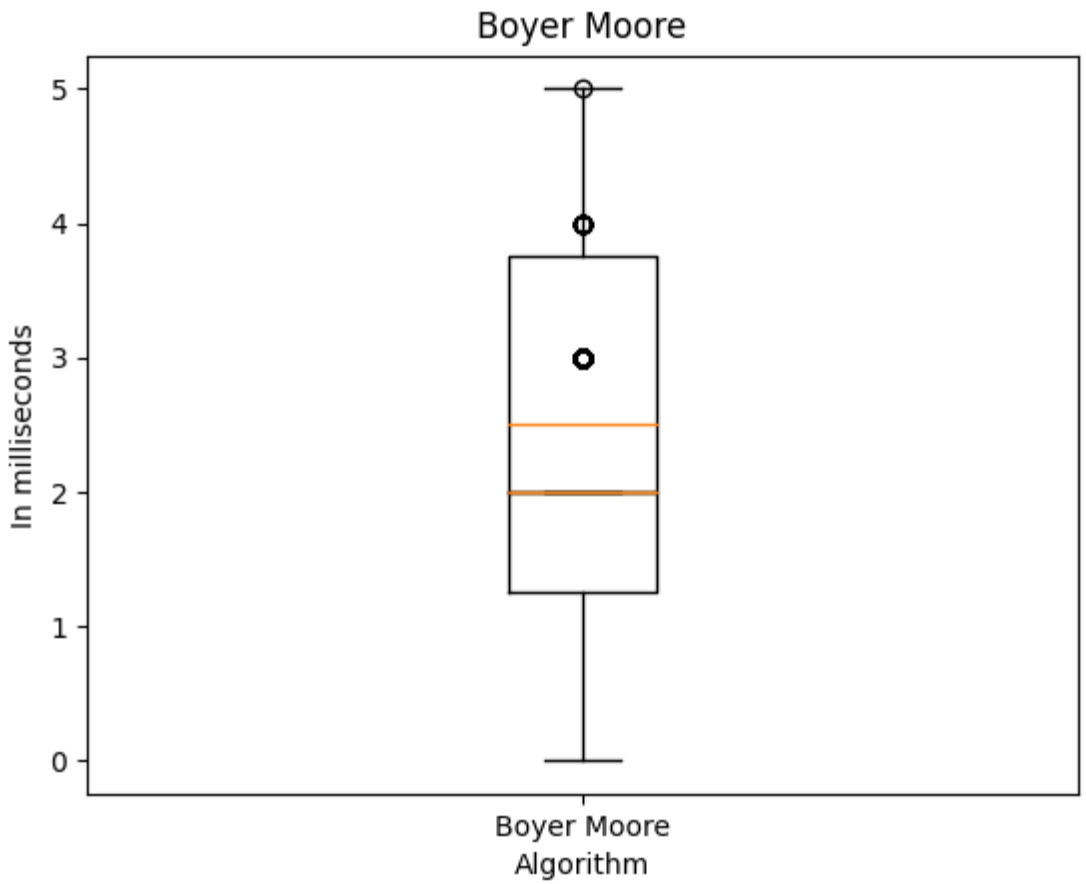


Pattern=Dundee (10,000 iterations)

1 Thread

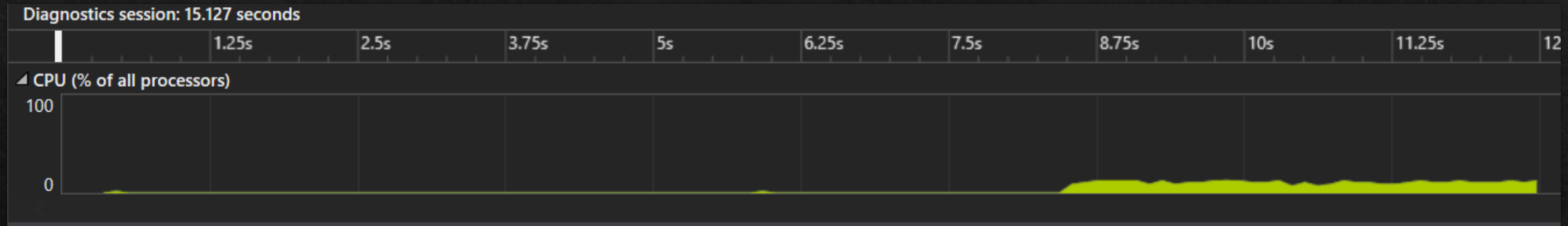


8 threads



Profiling – CPU usage

Searching Dundee with 1 thread.



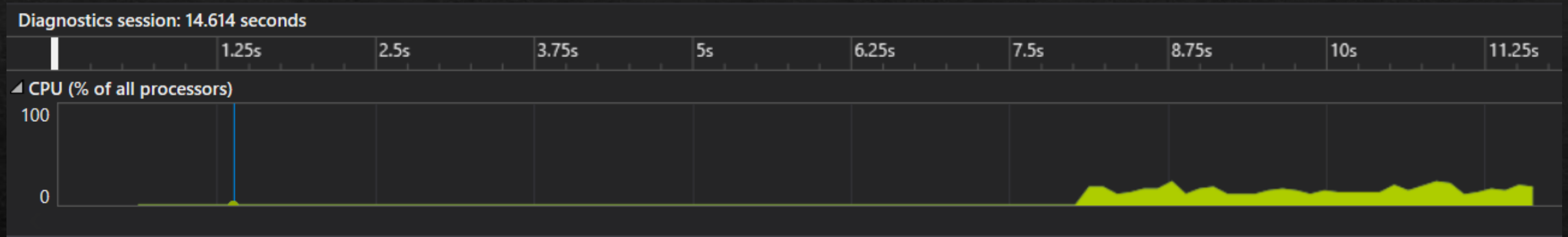
Oper

Top Functions

Function Name	Total CPU [unit, %]	Self CPU [unit, %]
stringSearch::run	3787 (93.46%)	3780 (93.29%)
[External Code]	4052 (100.00%)	248 (6.12%)
[External Call] vcruntime140.dll	21 (0.52%)	21 (0.52%)
[External Call] msvcp140.dll	2 (0.05%)	1 (0.02%)
std::list<__int64,std::allocator<__int64> >::list<__int64,std::allocator<__int64> >	2 (0.05%)	1 (0.02%)

Profiling – CPU usage

Searching Dundee with 8 threads.



Top Functions

Function Name	Total CPU [unit, %]	Self CPU [unit, %]
stringSearch::run	3036 (65.35%)	3031 (65.24%)
[External Code]	4646 (100.00%)	1603 (34.50%)
<lambda_9dff25d67048169b10ad8945de7c0246>::operator()	3245 (69.85%)	3 (0.06%)
main	424 (9.13%)	2 (0.04%)
test::~test	49 (1.05%)	2 (0.04%)

Effectiveness Of My Solution

- My objective was met. Parallelisation did speed up Boyer-Moore. More threads helped reduce time significantly as seen by the box plots.
- Task farm method very suitable when parallelising Boyer-Moore.
- However, searching smaller words (the) actually did not affect performance as much. Running 8 threads searching smaller words seemed to cut the time boyer-moore has to search in half.
- So, parallelisation did help in searching smaller words. Which was not what I expected.

Any questions