Pthreads and openCilk models

A comparison of the two parallel programming frameworks

Vasilis Kyriafinis, 9797

Aristotle University Of Thessaloniki
Parallel and Distributed Systems
October 2022

Contents

Introduction

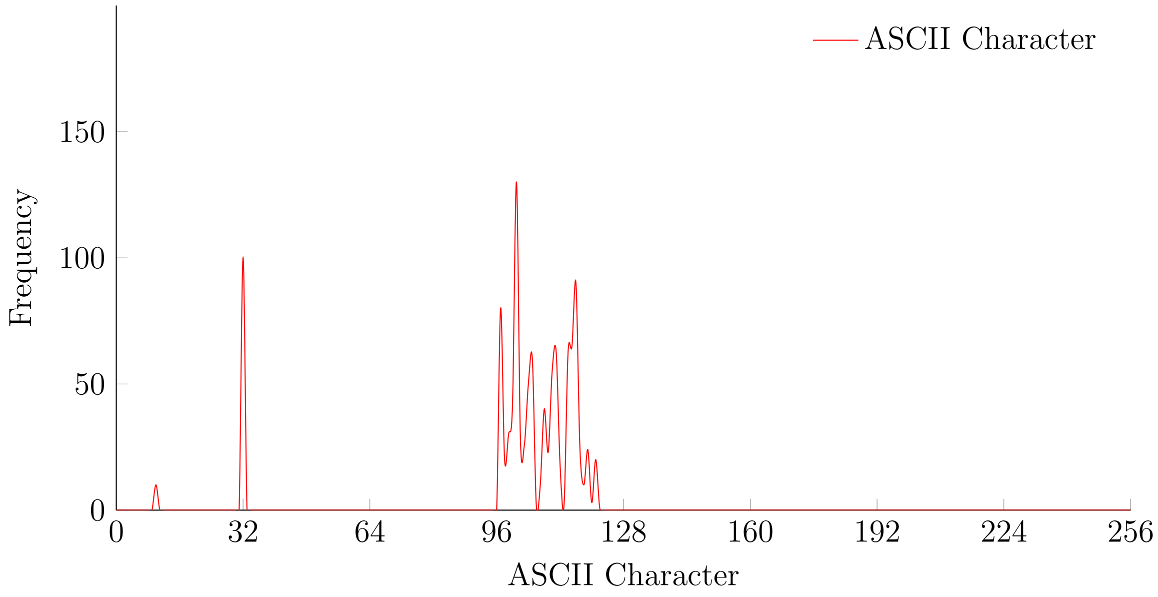
Assignment description

The main point of focus of this project is to compare the pthreads and openCilk multi-threading systems through a real life application. The main points of comparison are functionality, ease of use and performance, of the two multi-threading systems. The project uses C++ in order to be compatible with the uint256_t library but it is developed using structured programming logic. The code for this assignment can be found in this GitHub repository.

The real life problem

The problem used for this comparison is file compression using the Huffman algorithm. The Huffman algorithm is a lossless data compression algorithm. It is a variable length code algorithm, meaning that the length of the code is not fixed. The algorithm is based on the frequency of the characters in the file. The more frequent a character is, the shorter the code for that character will be. The algorithm performs best when the distributions of the frequencies is not uniform. For this reason the best compression is achieved on text files with natural language.

The natural language uses only part of the 8bit ASCII code. A typical frequency distribution of the characters in a text file is shown in the figure below.



Frequency distribution of characters in a text file

It is clear that the most frequent characters are the letters of the alphabet. Using the Huffman coding the symbols for the most frequent characters are less than the original 8bits of the ASCII code. Replacing the characters with those symbols will result in a smaller file size.

Huffman coding significance

The huffman algorithm is widely used in all the mainstream compression formats. It is especially effective in the compression of images, text and audio files.

It provides lossless compression, meaning that the original file can be reconstructed from the compressed file. These reasons have lead in the widespread use of the huffman algorithm. Because Huffman algorithm is a core element of many compression algorithms, speeding it up is very desirable, as many application will benefit from it.

State of the Art

Previous papers

Regarding the main focus point of this assignment there are plenty of papers comparing the performance of different threading programming models. Scott R. Taylor et al. [1] compared the performance of cilk, pthreads and Java threads. Their approach was to parallelize the fibonacci algorithm and to compare the performance of the different threading models. Their methodology was a time comparison between the three different models. They found that the cilk model was the fastest one and the Java threads model was the slowest one.

On a more comprehensive level, Ensar Ajkunic et al. [2] compared the performance of five different parallel programming models. Pthread, OpenMP, Threading Building Blocks, Cilk++ and MPI. Their approach was similar to the one of Taylor et al. [1] They compared the execution time of the algorithm written in the 5 models. The key difference was that they used matrix multiplication as the benchmarking algorithm. Their results also included speed up trends compared to the sequential implementation.

At last but not least Solmaz Salehian et al. [3] compared the performance of multiple threading models using multiple algorithms. The methodology was slightly different from the previous two papers. Along with the execution time they also compared the ability of the models to scale on multiple cores and the performance gains from using more physical cores.

Objective and Methodology

Objective

The main focus of this assignment is to compare the pthreads and openCilk models on multiple aspects. The most important is the performance of the benchmarking algorithm. The algorithm used as mentioned before is Huffman encoding. The sequential version of the algorithm was coded from scratch and can be found on the accompanying Github repository along with the parallel implementations.

Methodology

All of the test will be performed on the Ryzen 7 5700G, which is an 8 core 16 thread CPU. The variables of the tests will be the number of threads used and the size of the input file. All other variables are kept constant in a way that other components, such as disk read and write speeds, don't interfere with the results. The input file will be a text file containing lorem ipsum which produces character frequency distribution as shown in the introduction of this document. The results will include an absolute time comparison and a speed up trend compared to the sequential implementation.

Key differentiation

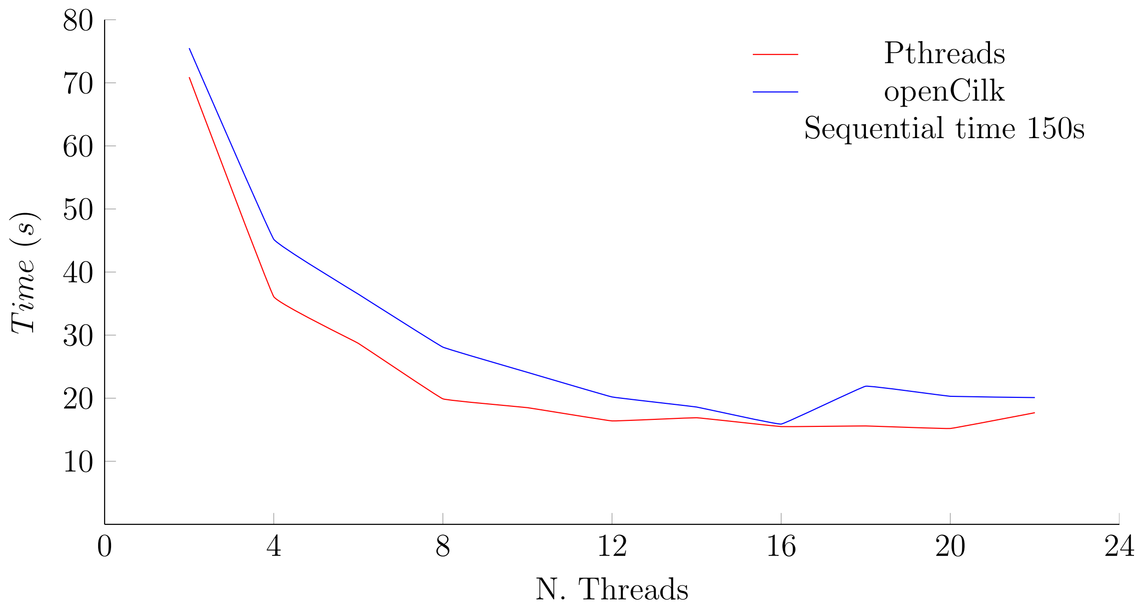
The main differentiation of this assignment for the previous papers is that it will also compare the ease of use of the two models as well as the functionality the two frameworks provide. Those aspects are inherently subjective and will be discussed in the results section.

Results

The results of the test run are presented bellow. Every test was averaged on multiple runs to reduce the variance of the results and to avoid the cold cache phenomenon.

Fixed input size

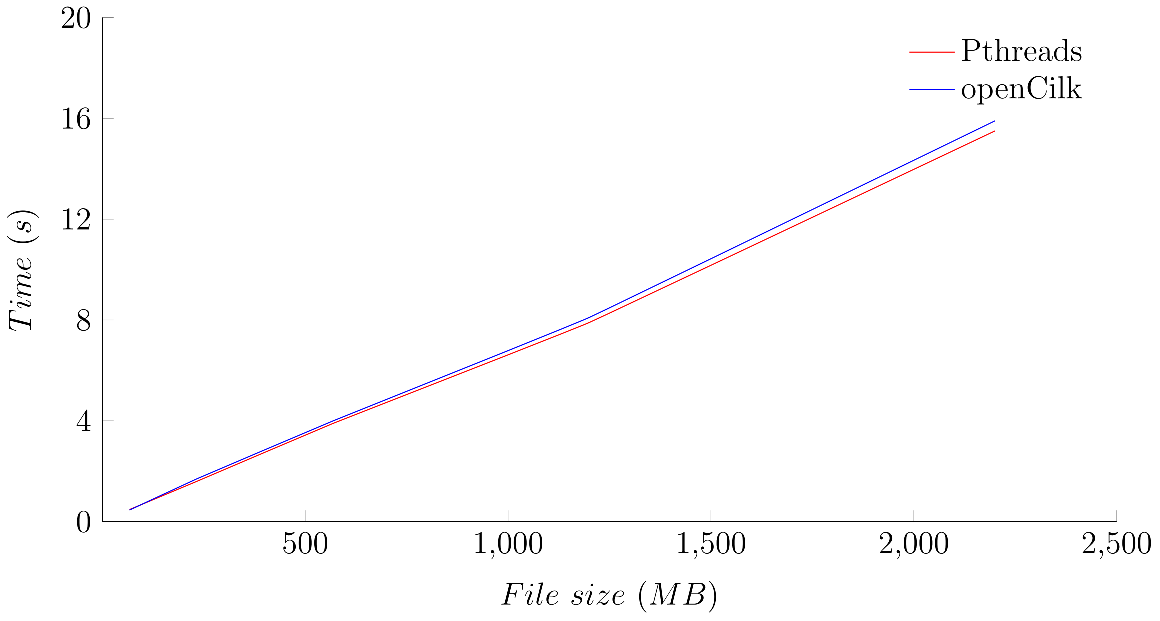
The first test run is with a fixed input size file of 2.2GB. The variable of this test is the number of threads opened by the program. The pthread implementation is slightly faster and the best performance is achieved for 16 threads. For more than 16 threads the performance is slowly dropping.



Time vs N. Threads for 2.2GB input

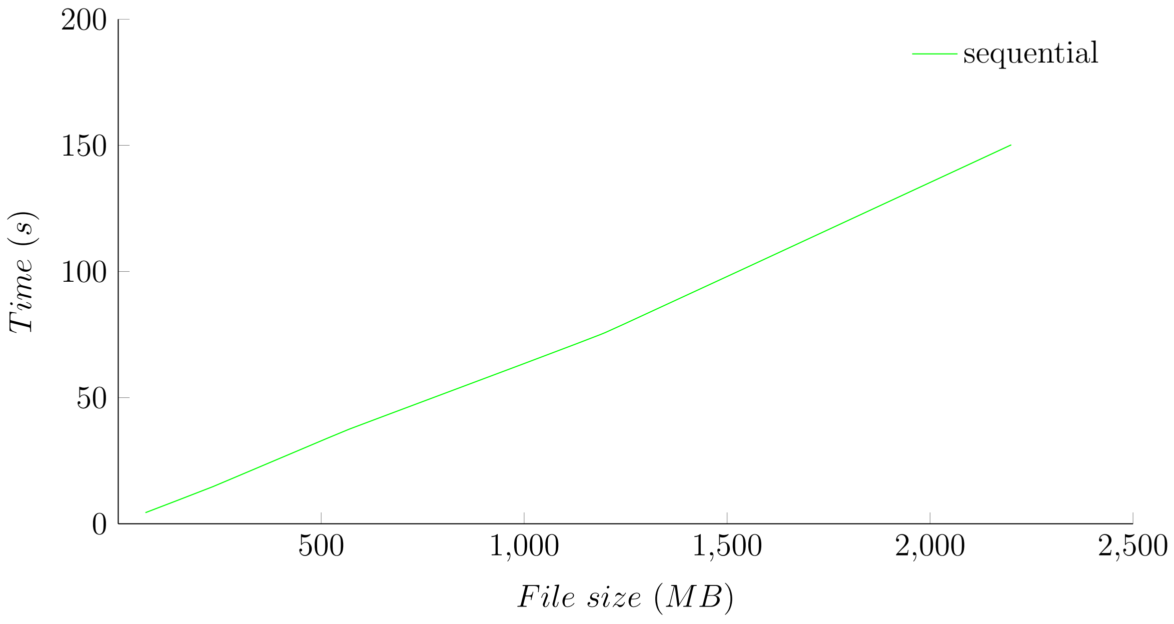
Fixed number of threads

The testing methodology is the same with the previous test, but this time the number of threads is fixed to 16. The variable for this test is the size of the input file. From the graph bellow, it can be seen that both models scale linearly with the input size and the performance is almost the same.



Time vs File size for 16 threads

This behavior is also present in the sequential algorithm, with the only difference being the execution time. This shows that both of the parallel implementations have the same time complexity with regard to the input size.

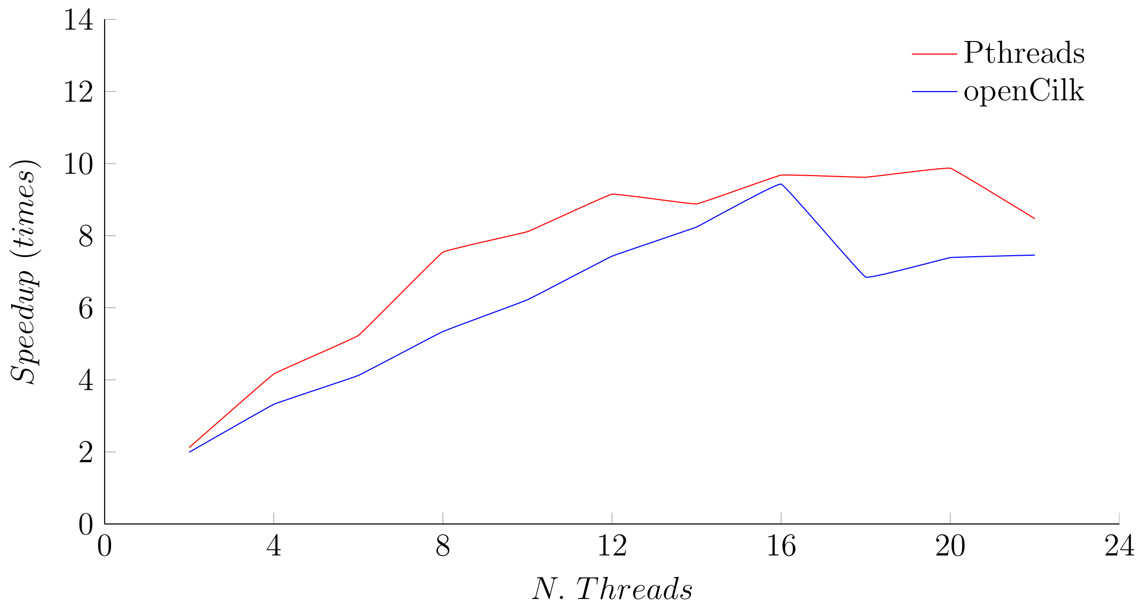


Time vs File size sequential

Speedup

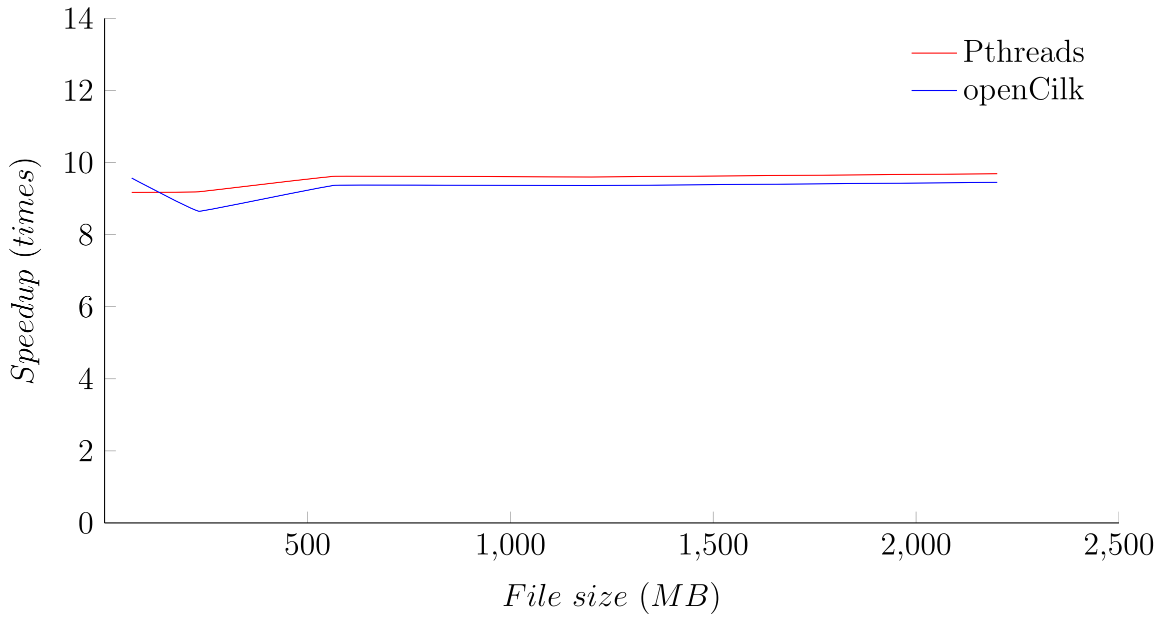
The speedup is the ratio between the execution time of the sequential algorithm and the execution time of the parallel algorithm. The speedup is a measure of the efficiency of the parallel algorithm. The speedup of the algorithm can be observed with a fixed number of threads and a fixed input size.

For the fixed input size test initially, the speedup is almost linear with the number of threads. Around 16 threads the speedup becomes almost constant. This is due to the fact that the CPU has 16 threads.



Speedup vs N. Threads for 2.2GB input

For the fixed number of threads test the speedup is constant. As mentioned before, this is due to the linear scaling of the algorithm with the input size.



Speedup vs File size for 16 threads

Functionality

Pthreads functionality

The pthreads framework provide maximum functionality. Everything is possible. This complexity though comes with a price. The pthreads framework is complex and requires a lot of work on the part of the programmer. The programmer on the other hand has maximum control over the program. If designed correctly the program can be very efficient and achieve great performance.

The pthreads framework is a well established library with plenty of documentation, tutorials and material available online. This only increases it's popularity and makes it a very good choice for a parallel programming framework.

openCilk functionality

The openCilk framework is a relatively new framework. It is a set of compiler extensions that allow the programmer to write parallel programs in a sequential manner. The complexity of designing the parallel program is hidden from the programmer behind the compiler. The side effect of this is that the programmer has less control over the program.

For the moment the openCilk framework has not achieved critical mass. It is slowly gaining popularity but the available documentation and tutorials are not very extensive. As a result it is can be tricky to get a good understanding of the framework.

Nevertheless, the ease of use of the openCilk framework makes it a very good choice for fast and easy parallelization of sequential algorithms. Finaly there are officialy supported utilities that help with the design of the parallel code.

Conclusions

Comparing the results of the testing revealed that the performance of the openCilk library is very close to that of the Pthreads. In addition openCilk is easier to use and significantly faster to implement compared to the Pthreads library. The work stealing architecture of the openCilk makes it very competitive with all the other parallel programming models.

This assignment did not include multiCilk in the comparison. MultiCilk is the combination of the work stealing architecture (thread pools) with the POSIX locks and mutexes. The implementation of the algorithm used in this assignment did not require any inter-thread communication or thread synchronization, so the usage of multiCilk was not required.

Bibliography

[1] Scott R. Taylor Diane J. Cook Krishna M. Kavi David Levine. *A Comparison of Multithreading Implementations*. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.57.728&rep=rep1&type=pdf.

[2] Ensar Ajkunic Hana Fatkic Emina Omerovic Kristina Talic Novica Nosovic. *A Comparison of Five Parallel Programming Models for C++*. URL: https://www.researchgate.net/publication/261424700_A_comparison_of_five_parallel_programming_models_for_C.

[3] Solmaz Salehian Jiawen Liu Yonghong Yan. *Comparison of Threading Programming Models*. URL: https://par.nsf.gov/servlets/purl/10050480.