C++ Concurrency Exercise

Richard Kroesen (774056)
29-7-2024 Aarhus Denmark

1. Introduction

This is a small report for the concurrency exercise, where a couple of measurements as benchmarks were taken.

1.1 Methodology

The measurements are taken by using chrono library and precise time functions, for time stamps. The interval is determined by the difference between the before and after the function call.

The function calls are made inside GoogleTesting framework unit tests.

Finally, this data is manually entered into a spreadsheet and a graphical representation is made to visually express the results.

2. Performance Benchmark

This chapter is giving the results of the taken benchmark.

2.1 Single vs Multi Tasks – Numeric

Below is the graphical representation of the measurements results of the numeric benchmark. The benchmark was between single and 10 threads function implementation.

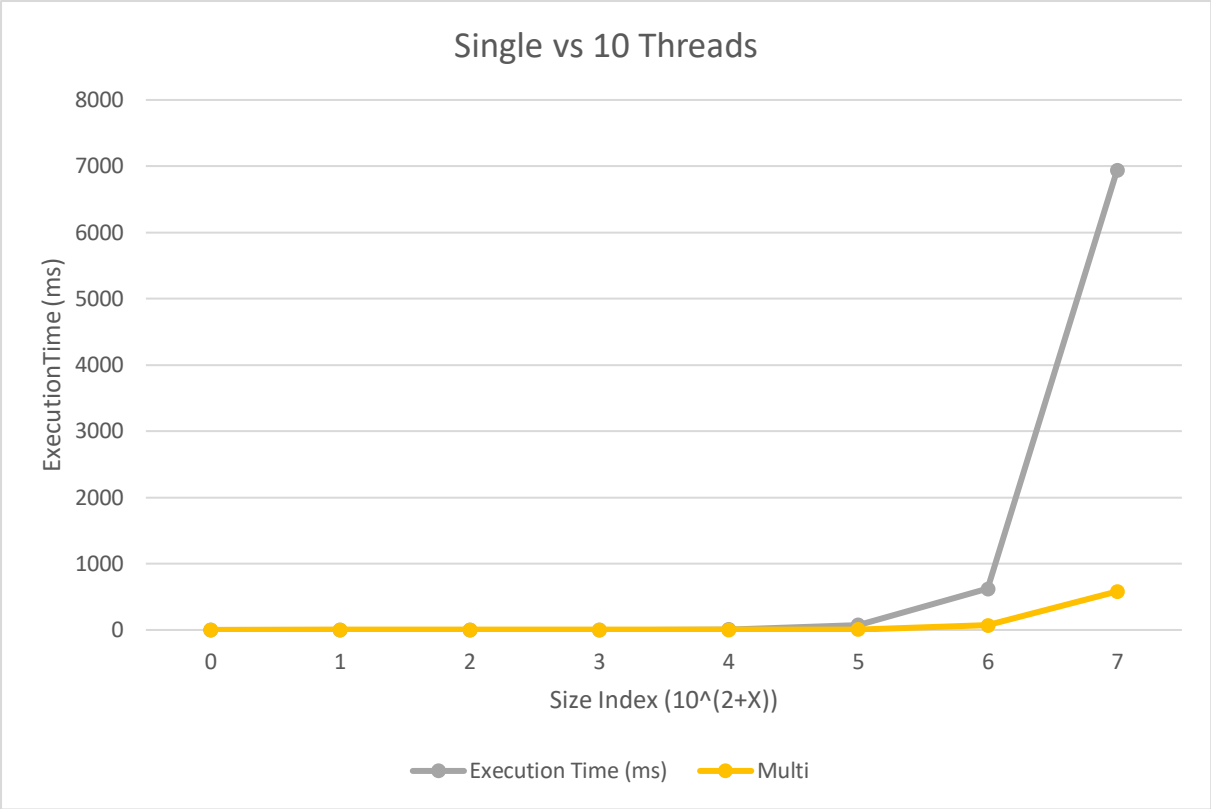

Figure 1 Results Benchmark Numberic - 1 vs. 10 tasks

The index is given to minimize the numeric representation. The actual value could be determined by filling the index into this formula: size = 10^(2+INDEX).

2.2 Single vs Multi Tasks – Strings

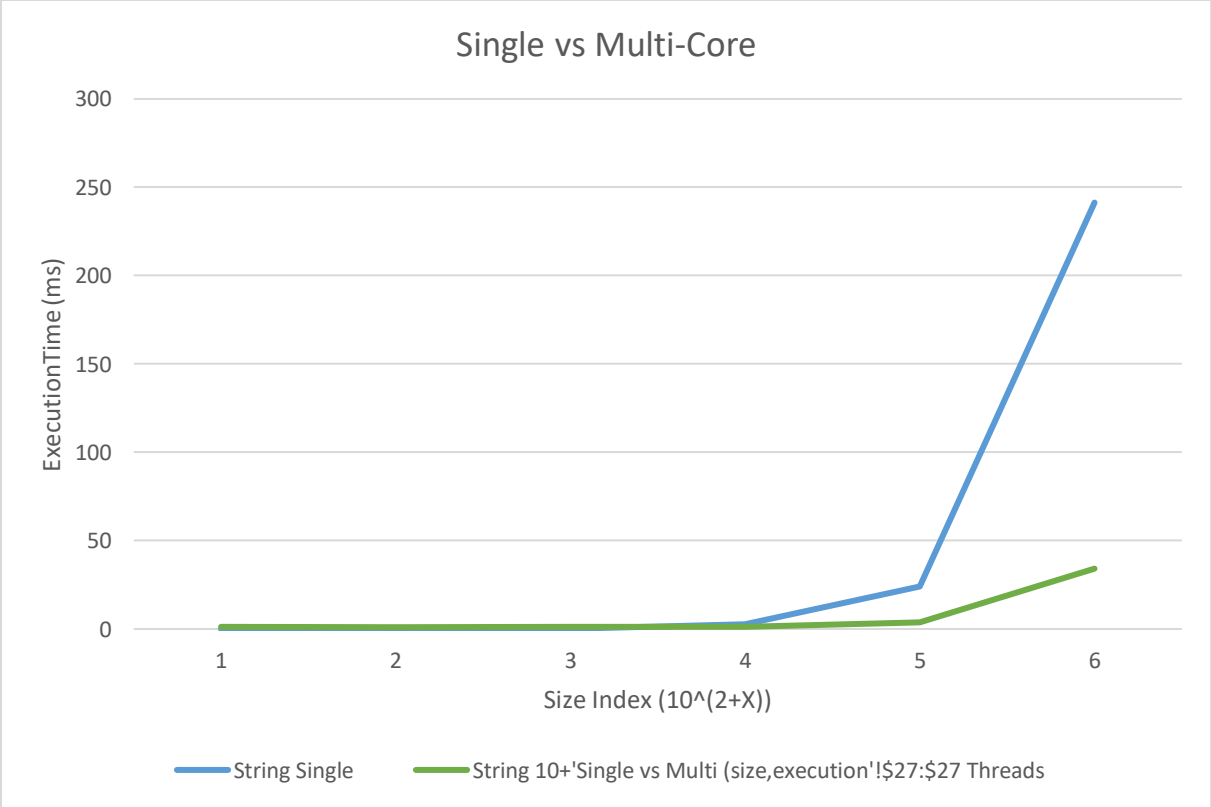

Figure 2 Results Benchmark strings - 1 vs. 10 tasks

2.3 The Amount of Tasks

In this measurement the number of threads was a variable instance. As well as the numeric function processing load.

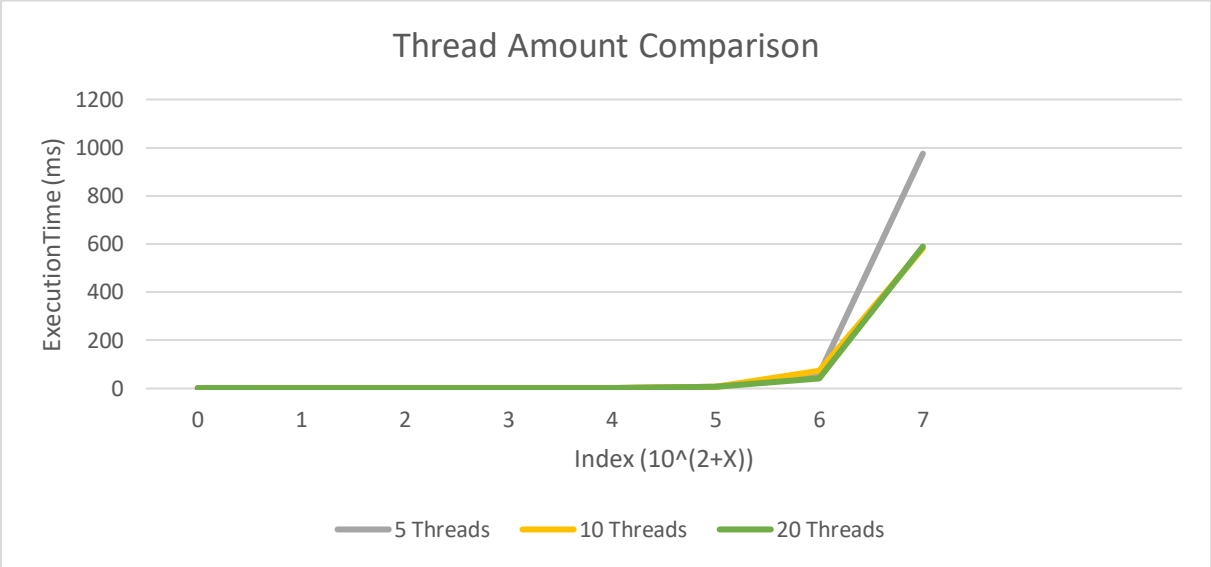

Figure 3 Amount of Tasks comparison

The index representation is still using the same format as discussed earlier.

2.4 Trade-off Concurrency

From the performed benchmark it could be stated that from 1,000,000 it is beneficial to start using multi-threading. In contrast, below this size the single operation is quite faster. For an enhanced insight the current benchmark program could be improved to setup the threads before useage. This is going to result in an improved measurement and a more fair comparison between the multi and single thread benchmark.

Number of threads also is an important matter to taken into account. From the quick measurement it could be found that after 10 threads there is no impactful benefit, with respect to 20 threads.

3. Conclusion

The benchmark results offer valuable insights into the trade-offs between single-threaded and multi-threaded operations in data processing. The findings suggest that the benefits of concurrency become apparent when dealing with datasets of 1,000,000 elements or more. For smaller datasets, single-threaded operations prove to be more efficient.

The number of threads utilized also plays a crucial role in optimizing performance. The preliminary measurements indicate that performance gains plateaued after 10 threads, with minimal additional benefit observed when increasing to 20 threads. This suggests that there is an optimal thread count beyond which additional parallelism yields diminishing returns.

3.1 Discussion

It's important to note that these findings are based on current benchmark setup. To gain a more comprehensive understanding, future iterations of the benchmark could be enhanced by initializing threads prior to their use. This modification would likely result in more accurate measurements and a fairer comparison between multi-threaded and single-threaded performance.

Also the exclusion of threads setup should be fixed in the next iteration.

In conclusion, while multi-threading can offer significant performance improvements for large-scale data processing tasks, such as finding all the elements in the given array, is not a one-size-fits-all solution. Developers should carefully consider the size of their datasets and the potential overhead of thread management when deciding between single-threaded and multi-threaded approaches. Furthermore, finding the optimal number of threads for a given task can lead to the best balance between performance gains and resource utilization.