Portfolio Project Option #1

Brady Chin

Colorado State University Global

CSC507-2: Foundations of Operating Systems

Dr. Joseph Issa

March 9th, 2025

Portfolio Project Option #1

For this portfolio project, we started with two very large text files. Each had 1 billion lines of random numbers. We were to add up the respective numbers in each row and place the sums in a new file. The files being so large in size, it was not feasible to load all of them at once in memory, and hence concurrency had to be utilized in order to make the process easier. To address this problem, two optimization techniques were used: a multithread approach, where data blocks were processed simultaneously by multiple threads, and a parallel processing approach, where files were divided into 10 small files and processed in parallel. The first objective was to compare the performance of these techniques based on processing time, efficiency, and their capacity to handle large-scale data.

Methods

To optimize the processing of the large files, I used two approaches: multithreading and parallel processing. In the multithreaded approach, the program was made to process segments of the data concurrently using multiple threads. Each thread was responsible for adding a segment of the files, i.e., the corresponding lines of hugefile1.txt and hugefile2.txt. This allowed the program to make use of more than one CPU core, speeding up the calculation by computing many lines at once. In the parallel processing approach, I divided the original files into 10 smaller files per file and ran multiple threads at the same time to process each file pair. Such strategy aimed to additionally distribute the load between the CPU cores, executing independent sets of files in parallel to speed up the overall run.

Results

The result showed a clear difference in performance between the two approaches. The multithreaded approach was significantly faster, taking 2.46 seconds to complete the task. This is because the ability of multiple threads to run simultaneously on different parts of the data, thus reducing the total processing time. The parallel processing approach was significantly slower, taking 136.05 seconds to process. The deceleration is highly probable because of the added complexity of handling multiple threads and files. Although the solution in theory should have picked up some efficiency through parallelization of the job, synchronization overheads and potential file-handling and I/O bottlenecks led to reduced efficiency when compared to the more straightforward multithreaded solution.

Interpretation

The experiment revealed that the multithreaded approach performed better in optimizing the task of working on the large file. By splitting the work into smaller parts and allowing multiple threads to process them at the same time, the program could utilize the multiplicity of the CPU, achieving a significant saving in processing time. In contrast, the method of parallel processing, despite the hope that it would further spread out the workload by cutting the files into chunks, was unsuccessful. The additional overhead of handling multiple files and synchronizing parallel operations resulted in high overhead in the form of potential I/O bottlenecks and synchronization overhead that proved to be too large. This demonstrated that while parallel processing, in principle, can lead to improved performance, it should be handled cautiously to avoid inefficiencies. The multithreaded method, as less complicated and more focused, performed optimally in this instance, reflecting the importance of finding a balance between complexity and optimization when working with large sets of data.

How This Course Helped

The course was very helpful in applying both the multithreading and parallel-processing techniques. It laid a great foundation in concurrency, parallelism, and thread management, which permitted me to think of a solution that would divide work effectively and optimize performance. Understanding the basics of thread synchronization, as well as task parallelism, enabled me to write a multithreaded program that would make full use of all CPU cores. In addition, focusing on practical problem-solving extends to identifying the functional problems associated with concurrent file processing, such as orchestrating file interaction and dealing with I/O bottlenecking. These were some of the primary components that made the difference between the two approaches in performance. Overall, this course was mostly about optimization in real-world systems and equipped me with the right assortments and perspectives to deal with tasks like this one, which encompass more challenging data processing.

Conclusion

Generally, the experiment recognized the necessity of selecting the optimum optimization technique while processing large files. The multithreaded process proved to be the most efficient by making the most of multiple CPU cores to process segments of data simultaneously, reducing the processing time significantly. On the other hand, the parallel processing solution, even with the potential of speeding up the task, suffered from the cost of processing many files and synchronizing tasks and thus had less performance. The exercise again reminded us of how concurrency and parallelism can improve large-scale data processing but at the same time emphasized the necessity of efficient system resource planning and task management in order to produce optimal performance. The knowledge gained through this course was crucial to comprehend the implementation of these techniques effectively in day-to-day situations.

References