PROGRAMAÇÃO PARALELA E CONCORRENTE

REPORT

Mestrado em Engenharia Informática
2023

Ana Teixeira - fc56336

What were the parallelization strategies used? How and why were they implemented?

1. **KnapsackParallelize:** This strategy utilizes parallelization by using multiple concurrent threads to manage distinct steps within a generation. To maintain the required sequential execution of the four essential steps, each step is parallelized independently to prevent concurrent processes.

2. **KnapsackSynchronized:** This strategy, similar to the parallelize method, uses the ***synchronized*** keyword for concurrency management on ‘population' and ‘newPopulation’ arrays. This ensures that only one thread at a time can access the critical section of the code, preventing potential data corruption within these arrays in this multithreading environment.

3. **KnapsackPhaser:** In this strategy, Phaser is used to coordinate parallel execution of the four interdependent steps within each generation. These steps must be executed sequentially, meaning all threads need to complete its execution on step 1 to proceed to step 2, and so forth. Without Phaser, the concurrent execution of these steps could lead to potential errors and data inconsistencies.

It’s crucial to highlight that parallelizing the generations was not a viable option because each generation’s results depend on the outcome of the previous one. Attempting this method would introduce concurrency issues, giving an unordered execution and potentially leading to incorrect results.

The ***volatile*** keyword was not used as a parallel strategy because it ensures that the reference to the array is always up to date, but it doesn't guarantee that the array’s values are up to date across multiple threads. In this context, the primary goal was to protect all the values within the array with the keyword to prevent concurrency issues. Using the ***volatile*** keyword would not achieve this specific objective, so it was not necessary.

How much better was the performance in parallel, compared to the sequential version?

The results include a boxplot graph showing the execution times for each method across a different number of threads. Notably, the median execution time for the sequential method is higher than all the other method’s median, which means parallelized strategies have a faster performance.
For the sequential method, data typically varies between 1.6e11 nanoseconds and 1.8e11 nanoseconds. In contrast, the data variation remains consistently under 1.6e11 nanoseconds for the parallelized methods (excluding outlier analysis). It is possible to note that all strategies have some outliers. However, the outliers of the parallelized methods have notably smaller values compared to the sequential method.

Analyzing the Kruskal Test results in conjunction with the boxplot graph, it's possible to notice that all strategies produce p-values very close to zero. This indicates significant differences between each strategy and the sequential method, emphasizing the notable impact of parallelization on performance.

It's worth noting that all the strategies employed maintain correctness, as the final result of the last generation for each strategy is approximately 4174, consistent with the result obtained from the sequential method.

How was the performance in each method in parallel?

When comparing the three parallelization methods without considering the number of threads used, it is notable that the synchronized method stands out as the least efficient. Its median runtime is around 1.4e11 nanoseconds while parallelize and the phaser method's median have smaller values (around 1.2e11 nanoseconds). This inefficiency in the synchronized method is due to the use of the ***synchronized*** keyword, which can lead to inefficiency in parallel execution because it may cause threads to wait for the access on synchronized blocks, slowing down the program. In contrast, although the phaser method has more distant outliers, its data remains more constant and its median is slightly smaller when compared to the parallelize method, which leads to the phaser strategy being the most efficient.

How was the performance within the different number of threads?

In this project, the three parallelization strategies were executed using 2 and 4 threads, corresponding to the available 4 cores on the machine used to execute the program. When utilizing 4 threads for parallelization, the time each method took to run 30 times was shorter compared to using only 2 threads. Therefore, the most efficient parallelization strategy is the phaser using 4 threads. It is also possible to conclude that the more threads used in the parallelization process, the faster and more efficient the program will be.

What was the experimental setup?

To analyze all the data, a csv file was created to contain all the execution times for the sequential method and for each parallelization strategy as well as for each number of threads used. Each method executed 30 times, and the 'stats.py' file creates a boxplot graph to compare the execution times of each method and it has statistical tests to compare each parallel strategy with the sequential method. The 'Main.java' is designed to run this program in any machine with any number of cores, but the machine used to execute the program only had 4 numbers of cores, being the reason why all strategies are only executed with 2 and 4 threads.