Assignment #1: Genetic Algorithm

for the Knapsack Problem

Programação Paralela e Concorrente (2023/2024)

Author: Daniel Lopes – Fc56357

1. What were the parallelization strategies used? How did you implement them and why?

- First of all, I developed a parallelization library named "ParallelLibrary," as required in TP Exercise 01. This library includes a method, "doWorkParallel," that allows tasks to be executed across multiple threads concurrently. The method accepts an instance of the "ParallelizeWork" interface, which represents a task with two parameters: "start" and "end," indicating the range of iterations that a specific thread should perform. It also takes the number of iterations and the desired number of threads.
- I use the "ThreadLocalRandom" instead of "Random" when generating random numbers in a multi-threaded context, as recommended by the professor. "ThreadLocalRandom" ensures thread safety by providing each thread with its independent instance for random number generation, improving efficiency.
- I adopted two primary strategies for parallelization:

    1. Identifying the most time-consuming parts of the code and parallelizing them individually. This approach involved parallelizing functions such as "populateInitialPopulationRandomly()," "bestOfPopulation()" and each step of the "run" method separately. For "bestOfPopulation()" I used synchronization to prevent concurrent access to the best value.

    2. Parallelizing the "run" method as a whole to parallelize work within each generation. Instead of creating and destroying threads (which is a time-consuming operation) for each step, I initiated threads in each generation iteration to perform all the work within the "for" loops concurrently. A "Phaser" was employed, initialized with the number of threads, to coordinate synchronization points for threads. Using local auxiliary variables, I synchronized access to the "population" array when copying data back to the main variable. Notably, I chose not to parallelize the "bestOfPopulation()" method to avoid unnecessary overhead and context switching when utilizing all available processors.

The performance improvement I achieved in the transition from the first to the second strategy was quite significant and I will provide more references to explain its impact.

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

Upon analyzing the execution times, a noticeable improvement in performance becomes evident. Within the data folder, two box plots are available: the first presents the execution time values for each type of execution (Sequential, First Strategy with N Threads, Second Strategy with N Threads), (filename: box_plot_execution_times.webp), and the second box plot highlights the percentage of improvement compared to the Sequential execution (filename: box_plot_perfomance_improvement.webp).

1. First Strategy: With only 2 threads, the improvement in execution time was approximately 45%. These values illustrate a significant boost in performance as more threads are employed for parallelization. It's essential to note that the performance gain tends to decrease as the number of threads increases (4 Threads: 67%, 8 Threads: 76%, 16 Threads: 81%).

2. Second Strategy: Overall, parallelizing the run method as a whole and implementing Phaser for synchronization consistently delivered better performance when compared to the initial approach (First Strategy). The average percentage improvement is consistently positive. For instance, the Second Strategy with 2 Threads, compared to the Sequential execution, exhibited a 53% improvement in execution time. And compared to the first Strategy, as more threads are added, the performance improvements will range from 8% to 3%.

In addition to the previously mentioned box plots, I conducted several statistical tests to assess the significance of the performance improvements:

- Kruskal-Wallis Test: This test determines if there is a significant difference in execution times among various combinations of program versions and thread counts. The test yielded a p-value less than alpha (0.05), indicating a statistically significant difference among at least two groups in the overall dataset.
- Friedman Test: Used to compare more than two related samples, this test aims to identify statistically significant differences in execution times between different versions of the program, specifically those employing the second strategy with Phaser for synchronization and varying thread counts (2 Threads, 4 Threads, 8 Threads, and 16 Threads). Similar comparisons were made for the first strategy. In both cases, the p-values were less than alpha, signifying significant differences between the groups.
- Wilcoxon Signed-rank Test: This test was employed in two instances, one comparing the Sequential execution with the first strategy and the other comparing it with the second strategy. In both cases, the results indicated significant differences between the two groups.

In summary, the combination of box plots and the outcomes of these statistical tests leads to the conclusion that the first strategy significantly improved performance. However, the second strategy surpassed it, confirming the efficacy of minimizing the thread count, as initially hypothesized.

3. **What was your experimental setup? Include all relevant details.**

The best values of each generation were solely assessed through visual inspection on the console, lacking any method to verify the correctness of both versions of the strategy. For my experimental setup, I conducted a series of experiments in the Main class where I recorded 30 execution times. The results were stored in a CSV file with columns including "ExecutionIndex," "Sequential," "ParallelExecutionTime2Threads(ns)," "ParallelExecutionTime4Threads(ns)," "ParallelExecutionTime8Threads(ns)," "ParallelExecutionTime16Threads(ns)", "ParallelPhaserExecutionTimeWith2Threads", The 'N' in "ParallelExecutionTimeWithNThreads(ns)" represents the number of available processors divided by 2, starting from the maximum available (in my case, 16 threads). This approach allowed me to test the program with multiple threads, and the flexibility to adapt to different machine configurations.

Subsequently, I moved the CSV file to the data folder where I have a Python script that reads the data and generates box plots. Additionally, I performed the statistical tests mentioned earlier.