PCM – Assignment 1

Environment:

All tests were performed in the same environment, a computer equipped with an 8 core (all locked at 4.2 GHz), 16 threads AMD Ryzen 7 3700X. I used the latest version of Eclipse and the background processes were kept to a minimum.

Knapsack:

For the sorting algorithm I used a merge sort algorithm. Being a recursive algorithm I decided to parallelize it by implementing a Fork Join parallelization. The task will keep being divided in two till the size of the array goes below a certain value, in this tests the value used was 8192. Regarding granularity I tried to use the number of queued surplus tasks to determine when the task should be done sequentially but the returned value never goes over 8 and making the task divide when the queued surplus task count was above 7 it worsen the run time. The returned values of the number of queued tasks, in my tests, never returned a value above 2. With that being said, the only value defining granularity is the threshold value for the array size, any other variables were not used and not included in the if statement in line 24 of the class ParallelMergeSort so that the if statement would have less variables to evaluate.

Regarding the rest of the program, all for loop minus the loop that iterates on the generations, they were parallelized using IntStream. When compared with the creation of threads, the creation of tasks has a lower overhead making them the best choice in this situation. All usage of the class Random were replaced for the class ThreadLocalRandom.

On average the program managed a speed up of 4.27 times.

NBody:

For parallelizing the advance method of the NBody problem it was used threads given the fact it was required a controlled split of the array, and IntStream isn’t capable of that. Each thread starts at a specific point of the array and is required to go through the rest of the array calculating the impact of the bodies in each other. To guarantee synchronization between threads it was used a synchronized block when the thread is changing values of the second body that is being calculating on. The only moment the same Body is being used at the same time is when a thread is using it as the “main” body (variable iBody) and another thread is trying to change the values as a secondary body (variable body), so the synchronized block is only needed in those three lines of code.

As for the loop that updates the geographical coordinates of each body it was used a IntStream since it has a lower overhead than threads and the order in which the values are updated have no importance.

On average the program managed a speed up of 2.06 times.