Software Engineering Coursework

Pavan Pinnaka
Second Year
Imperial College London

1 SECTION 1:

1.1 How long does it take to run?

Since the function which is being mapped creates a 2 second delay (2000 ms), and since each element in the list is being computed sequentially, it will take the (number of elements in the list * 2) seconds for all the elements in the list to be mapped. Which translates to 8 seconds in this case since there are 4 elements.

For the second part, I created a new class called Command which implements Callable. This class takes in a function and the argument of the function for its constructor and stores these values in its private fields. It only has one function (call() which has to be implemented since the class implements Callable). The function call() just returns the result of applying the function with the value. I had to do this because, I needed a way of implementing the call() method somewhere since when I submit a new class to the executor, it will look for the call() method which will be used. The executor then returns a new Future class instance which contains the value returned by the call method.

1.2 How long does it take to run after changing the map method to execute commands in parallel? State the factors that affect the runtime and also the limits.

Now that all the values are being mapped in parallel using the ExecutorService, it should take approximately 2 seconds to map the function over the entire list. It won't be exactly 2 seconds since there will be a slight overhead while adding the commands to the futureList and also adding the values of the futureList to the result list in the map function. Normally this overhead delay isn't really noticable if the number of elements in the MappableList are small. The obvious limiting factor to this system is the number of ThreadPools we create when declaring the Executors. The answer provided assumes that this number is equal to the number of elements in the MappableList, which is why it only takes the time to map one element to map the entire list. However, if we reduce the number of ThreadPools available, they Commands will be queued if the ThreadPools are full and only map n elements in parallel, where n is the number of ThreadPools. And obviously setting the ThreadPools size really high (greater than the number of elements) wouldn't help since the extra ThreadPolls will simply wont be used.

1.3 Try creating a function that actually does take some time to compute, rather than the artificial delay we introduced above. What are the results now?

To simulate a scenario where a function doesn't always take the same amount of delay to execute, I created a new function which is similar to the square function, however instead of using a constant value of 2000 ms for the delay, I used a random number generator to make the function generate random delays each time it is executed. When I mapped this new function on the MappableList, the results where quite interesting, as the order at which the elements were mapped, directly corresponded to the time the function took to execute (ie. elements being mapped from functions with low delays being executed first). All the elements were still mapped in parallel, meaning that the time taken to map the whole list will the maximum time a function took to compute the result. Essentially the order of

execution became deterministic as soon the function had different times of execution, whereas when I mapped the square function, there was non-determinacy as to which element will be mapped first since the function always had a delay of 2000 ms.

2 EXTENSION:

2.1 What sort of speed-up can you achieve for reduce when ran in parallel?

Running the reduce function in parallel essentially maps all the pairs in the list in parallel splitting the task into several levels untils its reduced down to a single element. Therefore the total time taken will be less than $\lceil (n/2) \rceil * (t)$ if n is an odd value, where n is the number of elements in the list and t is the time it takes a binary function to compute the value. And less than $(n/2) * t$ if the number of elements is an even value. It is important to note that reducing a list down in parallel with a function which isn't assosiative, results in an invalid answer since the order in which the pair of elements will be reduced down is non-deterministic at each stage of the reduce.