Compact Programming Course: Home Assignment 3

1. Comparison of concurrency models

    a. Parallel workers:
        i. Advantages:
            1. **Easy to understand**. To increase the parallelization level of the application you just add more workers.
            2. **Example**: implementation of a web crawler, you could crawl a certain number of pages with different numbers of workers and see which number gives the shortest total crawl time (meaning the highest performance).
        ii. Disadvantages:
            1. **Complex Management of Shared State**: The threads need to access the shared data in a way that makes sure that changes by one thread are visible to the others (pushed to main memory and not just stuck in the CPU cache of the CPU executing the thread). Threads need to avoid race conditions, deadlock and many other shared state concurrency problems. part of the parallelization is lost when threads are waiting for each other when accessing the shared data structures.
            2. **Slow Speeds**. workers must re-read the state every time they need it, to make sure they are working on the latest copy. Re-reading data every time you need it can get slow. Especially if the state is stored in an external database.
            3. **Job Ordering is Nondeterministic**. There is no way to guarantee which jobs are executed first or last. Job A may be given to a worker before job B, yet job B may be executed before job A.

    b. Assembly Line:
        i. Advantages:
            1. **No Shared State**. workers share no state with other workers means that they can be implemented without having to think about all the concurrency problems that may arise from concurrent access to shared state.
            2. **Faster Speeds**. workers know that no other threads modify their data, the workers can keep the data they need to operate in memory, only writing changes back the eventual external storage systems. Therefore, faster than a stateless worker.
            3. **Better Hardware Conformity.** conforms better with how the underlying hardware works. can usually create more optimized data structures and algorithms. data is cached in memory there is also a higher probability that this data is also cached in

the CPU cache of the CPU executing the thread. This makes accessing cached data even faster.

4. Job Ordering is Possible.

ii. Disadvantages:
   1. **Hard to understand**. execution of a job is often spread out over multiple workers, and thus over multiple classes in your project. Thus, it becomes harder to see exactly what code is being executed for a given job.
   2. **Hard to code**. Having code with many nested callback handlers may result in what some developer call *callback hell*.

c. Functional Parallelism
   i. Advantages:
      1. The basic idea of functional parallelism is that you implement your program using function calls. Functions can be seen as "agents" or "actors" that send messages to each other, just like in the assembly line concurrency model (AKA reactive or event driven systems). When one function calls another, that is similar to sending a message.
   ii. Disadvantages:
      1. The hard part about functional parallelism is knowing which function calls to parallelize. Coordinating function calls across CPUs comes with an overhead. The unit of work completed by a function needs to be of a certain size to be worth this overhead. If the function calls are very small, attempting to parallelize them may be slower than a single threaded, single CPU execution.

2. Concurrency vs Parallelism

   Concurrency means that an application is making progress on more than one task - at the same time or at least seemingly at the same time (concurrently).

   If the computer only has one CPU the application may not make progress on more than one task at exactly the same time, but more than one task is in progress at a time inside the application. To make progress on more than one task concurrently the CPU switches between the different tasks during execution.

   The term parallelism means that an application splits its tasks up into smaller subtasks which can be processed in parallel, for instance on multiple CPUs at the exact same time. Thus, parallelism does not refer to the same execution model as parallel concurrent execution - even if they may look similar on the surface.

To achieve true parallelism your application must have more than one thread running - and each thread must run on separate CPUs / CPU cores / graphics card GPU cores or similar.

3. Blocking Concurrency Algorithms and Non-blocking Concurrency Algorithms

    a. Blocking Concurrency Algorithms
    A blocking concurrency algorithm is an algorithm which either: Performs the action requested by the thread or Blocks the thread until the action can be performed safely

    b. Non-blocking Concurrency Algorithms
    A non-blocking concurrency algorithm is an algorithm which either: Performs the action requested by the thread or Notifies the requesting thread that the action could not be performed

The main difference between blocking and non-blocking algorithms lies in the second step of their behavior as described in the above two sections. In other words, the difference lies in what the blocking and non-blocking algorithms do when the requested action cannot be performed:
Blocking algorithms block the thread until the requested action can be performed.
Non-blocking algorithms notify the thread requesting the action that the action cannot be performed.

References:

https://jenkov.com/tutorials/java-concurrency/index.html
https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html