Java Concurrency

Concurrency vs. Parallelism

- Concurrency vs Parallelism Tutorial Video
- Concurrency
- Parallel Execution

- Concurrent, Not Parallel
- Parallel, Not Concurrent
- Neither Concurrent Nor Parallel
- Concurrent and Parallel

Jakob Jenkov
Last update: 2020-11-17

The terms *concurrency* and *parallelism* are often used in relation to multithreaded programs. At first it may seem as if concurrency and parallelism may be referring to the same concepts. However, concurrency and parallelism actually have different meanings. In this concurrency vs. parallelism tutorial I will explain what these concepts mean.

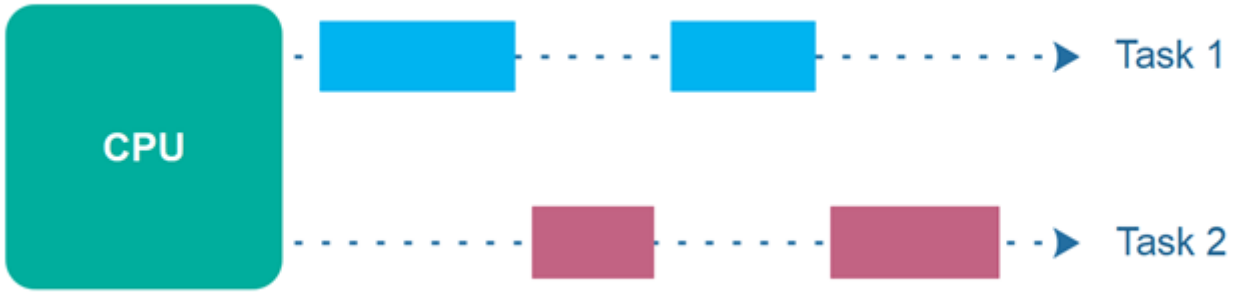
Just to be clear, in this text I look at concurrency and parallelism within a single application - a single process. Not among multiple applications, processes or computers.

Concurrency vs Parallelism Tutorial Video

If you prefer video, I have a video version of this tutorial here: **Concurrency vs Parallelism Tutorial Video**

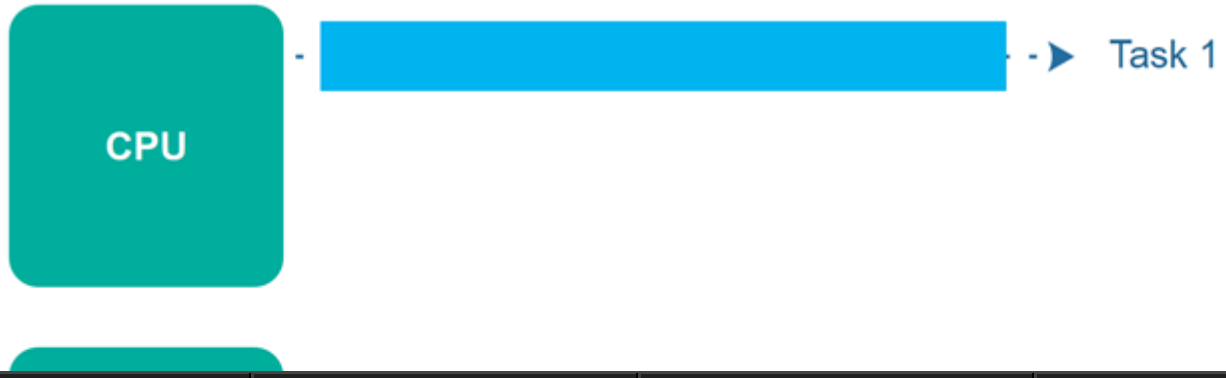
Concurrency

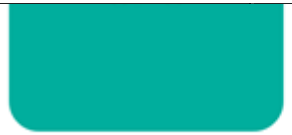
If the computer only has one CPU the application may not make progress on more than one task at *exactly the same time*, but more than one task is in progress at a time inside the application. To make progress on more than one task concurrently the CPU switches between the different tasks during execution.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



Parallel Execution

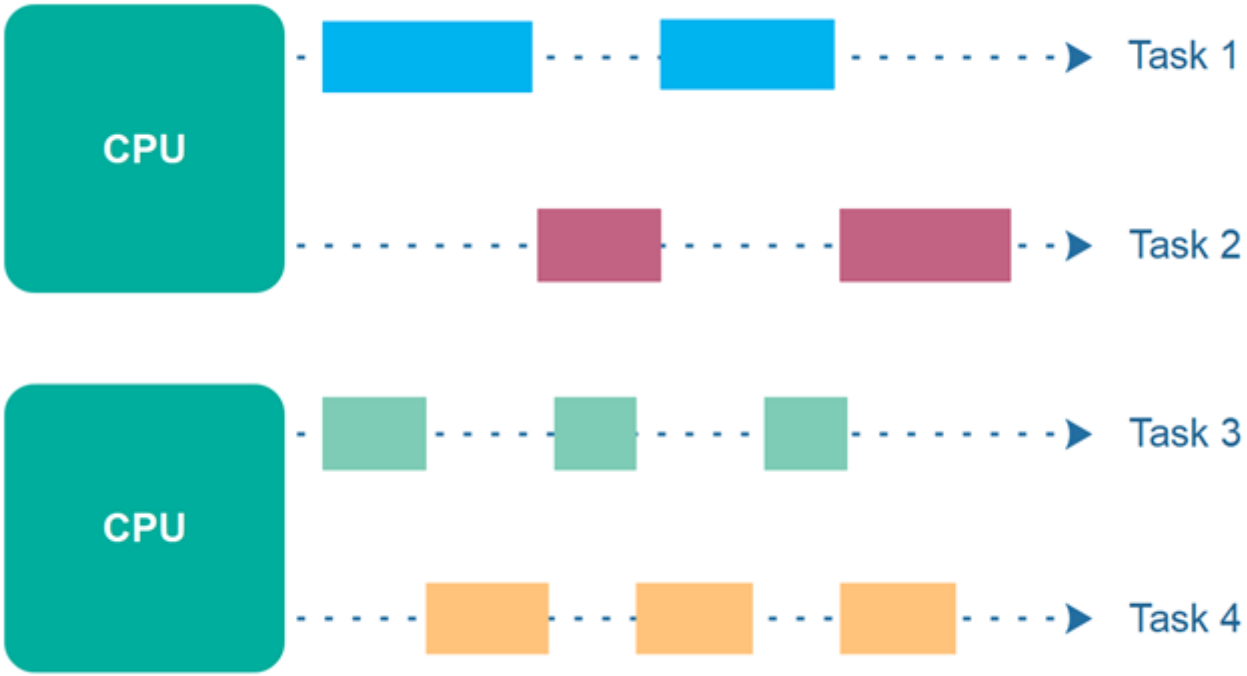
Parallel execution is when a computer has more than one CPU or CPU core, and makes progress on more than one task simultaneously. However, *parallel execution* is not referring to the same phenomenon as *parallelism*. I will get back to parallelism later. Parallel execution is illustrated below:





Parallel Concurrent Execution

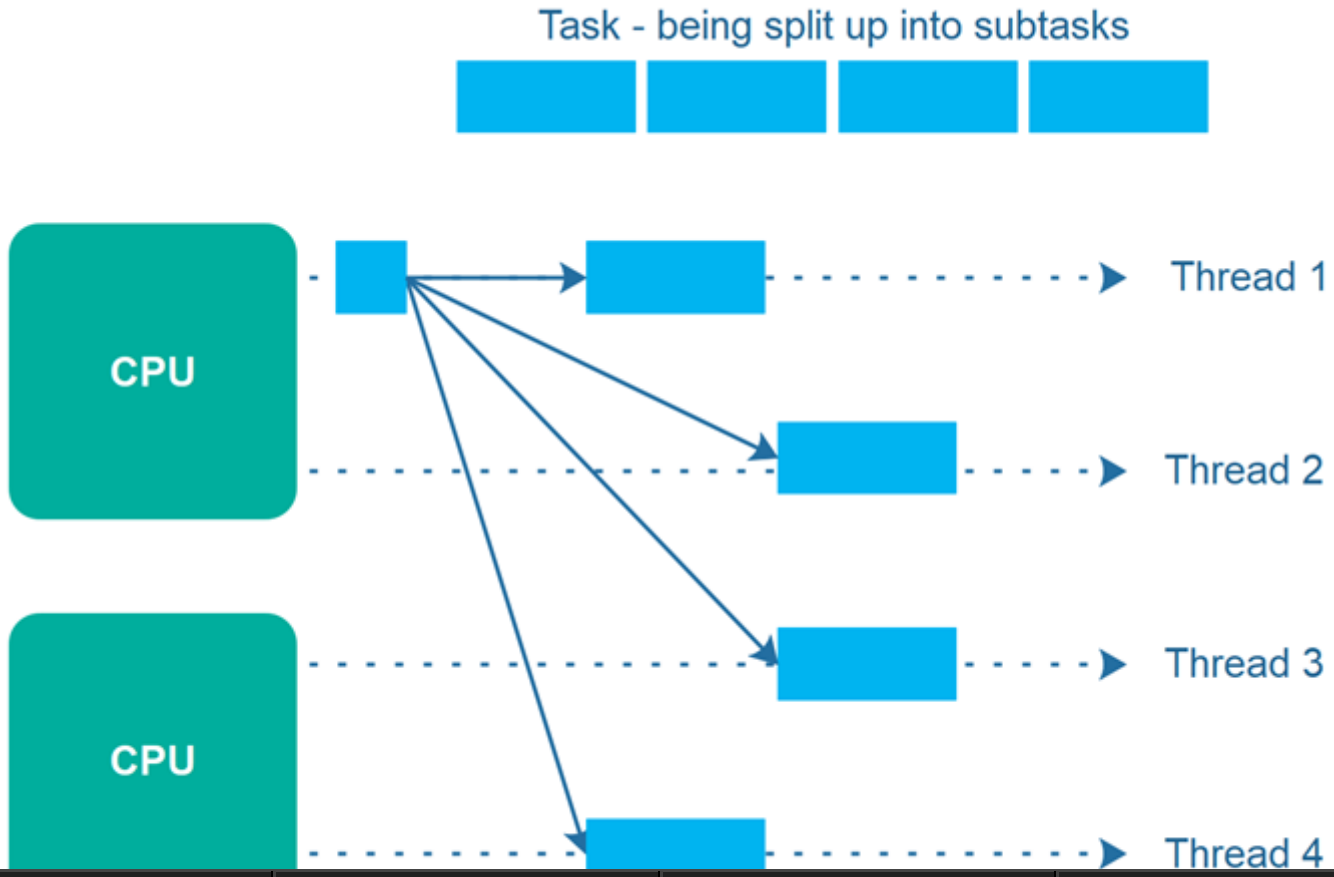
It is possible to have parallel concurrent execution, where threads are distributed among multiple CPUs. Thus, the threads executed on the same CPU are executed concurrently, whereas threads executed on different CPUs are executed in parallel. The diagram below illustrates parallel concurrent execution.



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.

The diagram below illustrates a bigger task which is being split up into 4 subtasks. These 4 subtasks are being executed by 4 different threads, which run on 2 different CPUs. This means, that parts of these subtasks are executed concurrently (those executed on the same CPU), and parts are executed in parallel (those executed on different CPUs).



If instead the 4 subtasks were executed by 4 threads running on each their own CPU (4 CPUs in total), then the task execution would have been fully parallel. However, it is not always easy to break a task into exactly as many subtasks as the number of CPUs available. Often, it is easier to break a task into a number of subtasks which fit naturally with the task at hand, and then let the thread scheduler take care of distributing the threads among the available CPUs.

Concurrency and Parallelism Combinations

To recap, *concurrency* refers to how a single CPU can make progress on multiple tasks seemingly at the same time (AKA concurrently).

Parallelism on the other hand, is related to how an application can parallelize the execution of a single task - typically by splitting the task up into subtasks which can be completed in parallel.

These two execution styles can be combined within the same application. I will cover some of these combinations below.

Concurrent, Not Parallel

An application can be concurrent, but not parallel. This means that it makes progress on more than one task seemingly at the same time (concurrently), but the application switches between making progress on each of the tasks - until the tasks are completed. There is no true parallel execution of tasks going in parallel threads / CPUs.

Parallel, Not Concurrent

An application can also be parallel but not concurrent. This means that the application only works on one task at a time, and this task is broken down into subtasks which can be processed in parallel. However, each task (+ subtask) is completed before the next task is split up and executed in parallel.

Neither Concurrent Nor Parallel

Additionally, an application can be neither concurrent nor parallel. This means that it works on only one task at a time, and the task is never broken down into subtasks for parallel execution. This could be the case for small command line applications where it only has a single job which is too small to make sense to parallelize.

Concurrent and Parallel

Finally, an application can also be both concurrent and parallel in two ways:

The second way is that the application both works on multiple tasks concurrently, and also breaks each task down into subtasks for parallel execution. However, some of the benefits of concurrency and parallelism may be lost in this scenario, as the CPUs in the computer are already kept reasonably busy with either concurrency or parallelism alone. Combining it may lead to only a small performance gain or even performance loss. Make sure you analyze and measure before you adopt a concurrent parallel model blindly.

Next: Single-threaded Concurrency

Tweet

Jakob Jenkov


7 + 2 Things I Love
About Java


False Sharing
in Java
Jakob Jenkov
JENKOV.COM


Java Concurrency +
Cache Coherence
Jakob Jenkov
JENKOV.COM


Compare and Swap
in Java
Jakob Jenkov
JENKOV.COM


Producer Consumer Pattern
in Java
Jakob Jenkov