Recently, I was watching a video about concurrency and parallelism. When I started explaining myself about this I ended up in other related concepts and nomenclatures such as Threads -> Multi-threaded and Single, Asynchronous and Synchronous. At a point, I was confused with queries like:

How is concurrency related to parallelism?

What is synchronous and asynchronous execution?

How do you distinguish between concurrent & parallel?

How threads fit along with all these concepts?

Many of us sometimes get confused with such queries.

Hold on for a moment and I will try to answer the concurrency and parallelism queries and visualize the concepts.

Concurrency

Concurrency means that an application is making progress on more than one task at the same time (concurrently). Well, 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 being processed at a time inside the application. It does not completely finish one task before it begins the next.

let's take an example in real life: There’s a challenge that requires you to both eat a whole huge cake and sing a whole song. You’ll win if you’re the fastest who sings the whole song and finishes the cake. So the rule is that you sing and eat simultaneously, How you do that does not belong to the rule. You can eat the whole cake, then sing the whole song, or you can eat half a cake, then sing half a song, then do that again, etc.

Concurrency means executing multiple tasks at the same time but not necessarily simultaneously. There are two tasks executing concurrently, but those are run in a 1-core CPU, so the CPU will decide to run a task first and then the other task or run half a task and half another task, etc. Two tasks can start, run, and complete in overlapping time periods i.e Task-2 can start even before Task-1 gets completed. It all depends on the system architecture.

> *Concurrency means executing multiple tasks at the same time but not necessarily simultaneously.*

Parallelism

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.

Parallelism does not require two tasks to exist. It literally physically run parts of tasks OR multiple tasks, at the same time using the multi-core infrastructure of CPU, by assigning one core to each task or sub-task.

If we keep going with the same example as above, the rule is still singing and eating concurrently, but this time, you play in a team of two. You probably will eat and let your friend sing (because she sings better and you eat better). So this time, the two tasks are really executed simultaneously, and it's called *parallel*.

Parallelism requires hardware with multiple processing units, essentially. In single-core CPU, you may get concurrency but NOT parallelism. Parallelism is a specific kind of concurrency where tasks are really executed simultaneously.

What is the difference between parallel programming and concurrent programming?

Now let's list down remarkable differences between concurrency and parallelism.

For instance, The Art of Concurrency defines the difference as follows:

> A system is said to be *concurrent* if it can support two or more actions *in progress* at the same time. A system is said to be *parallel* if it can support two or more actions executing

simultaneously.

The key concept and difference between these definitions is the phrase “in progress.”

This definition says that, in concurrent systems, multiple actions can be *in progress*(may not be executed) at the same time. Meanwhile, multiple actions are simultaneously executed in parallel systems. In fact, concurrency and parallelism are conceptually overlapped to some degree, but “in progress” clearly makes them different.

Concurrency is about dealing with lots of things at once. Parallelism is about doing lots of things at once.

An application can be concurrent — but not parallel, which means that it processes more than one task at the same time, but no two tasks are executing at the same time instant.

An application can be parallel — but not concurrent, which means that it processes multiple sub-tasks of a task in multi-core CPU at the same time.

An application can be neither parallel — nor concurrent, which means that it processes all tasks one at a time, sequentially.

An application can be both parallel — and concurrent, which means that it processes multiple tasks concurrently in multi-core CPU at the same time.

Summary

I recommend using the term “parallel” when the simultaneous execution is assured or expected, and to use the term “concurrent” when it is uncertain or irrelevant if simultaneous execution will be employed.

I would, therefore, describe simulating a jet engine on multiple cores as parallel.

Say you want to *compress n text files* and generate a compressed file for each of them. You can have from 2 (up to n) threads that each handle compressing a subset of the files. When each thread is done, it’s just done, it doesn’t have to wait or do anything else. So, since different tasks are performed in an interleaved manner in “any arbitrary order” the program is concurrent but not parallel.

Distinguishing parallelism from concurrency is important to seek a fitting way to solve large scale problems, but they are considered interchangeable in reality. The definitions provided are tremendously valuable to figure out the very similar but different two paradigms. It is worth watching the video about concurrency and parallelism as this talks more deep with better example(in the context of GO)