Parallelism vs. Concurrency

From HaskellWiki

The term **Parallelism** refers to techniques to make programs faster by performing several computations at the same time. This requires hardware with multiple processing units. In many cases the sub-computations are of the same structure, but this is not necessary. Graphic computations on a GPU are parallelism. A key problem of parallelism is to reduce data dependencies in order to be able to perform computations on independent computation units with minimal communication between them. To this end, it can even be an advantage to do the same computation twice on different units.

The term **Concurrency** refers to techniques that make programs more usable. Concurrency can be implemented and is used a lot on single processing units, nonetheless it may benefit from multiple processing units with respect to speed. If an operating system is called a multi-tasking operating system, this is a synonym for supporting concurrency. If you can load multiple documents simultaneously in the tabs of your browser and you can still open menus and perform other actions, this is concurrency.

If you run distributed-net computations in the background while working with interactive applications in the foreground, that is concurrency. On the other hand, dividing a task into packets that can be computed via distributed-net clients is parallelism.

Contents

An anecdote from the good old Amiga days

An anecdote on how concurrency is useful, also without parallelism: Amiga computers were always advertised for their multi-tasking operating system. However DOS/Windows-3.1 users were never attracted by this advertisement since they argued that a single CPU cannot be made faster by performing several tasks in an interleaved way. They were right, but this was not the point: Multitasking avoids that the computer gets bored. Indeed in the eighties Amiga computers were considered great for raytracing. However the special graphics and sound hardware in Amiga computers could not help with raytracing. The important advantage was, that you could perform the graphics rendering concurrently to your daily work (office applications) without noticing the computation load of the raytracing. Multitasking just assigns the time between your keystrokes to the raytracer. However multitasking was not possible with most games, office software that eats all the memory or simply crashing applications. This leads to another confusing area: Error vs. Exception.

How to distinguish between Parallelism and Concurrency

- If you need `getNumCapabilities` in your program, then your are certainly programming parallelism.
- If your parallelising efforts make sense on a single processor machine, too, then you are certainly programming concurrency.

Warning

Not all programmers agree on the meaning of the terms 'parallelism' and 'concurrency'. They may define them in different ways or do not distinguish them at all.

See also

- Note of Simon Marlow in Haskell-Cafe (http://www.haskell.org/pipermail/haskell-cafe/2008-September/047312.html) about the distinction of parallelism and concurrency
- GHC mutterings on Parallelism /= Concurrency (http://ghcmutterings.wordpress.com/2009/10/06/parallelism-concurrency/)

Categories: Idioms | Parallel
