Concurrency in Go

Ian Harris

December 14, 2018

1 Go's Implementation of Concurrency

One of the big advantages of Go, is its implementation of concurrency. So, we'll talk a little bit right now about concurrency, what it is, and why it's useful, and how Go implements it, how there are built-in constructs in the language that make it easy to use concurrency.

1.1 Performance Limitations

So, I'm going to start by talking about performance limitations of computers.

1.1.1 Need for Speed

The reason for this is because a lot of the motivation for concurrency comes from the need for speed. A lot of the motivation, not all but a lot of it does. So, that's why I'm going to introduce these performance limits on machines and how concurrency can help you get around these performance limitations.

1.1.2 Moore's Law

So, Moore's law, so just to summarize, you probably heard of this law. But in case you have not, Moore's Law, it basically says, that the number of transistors on a chip doubles every 18 months. Now, this used to be the case, it is not the case anymore, recently. That has changed, but this used to be the case and so, because of this doubling of transistors, what happened would be, machines would speed up just. Because as the transistors got a little smaller and they would be closer to each other, you could increase the clock rate and so clock rates would just increase, increase, increase. I remember when I was in school, which was a while ago, you'd buy a machine and seriously a few months later there'd be another machine, same price, it was faster and it was frustrating. So, these clock rates were just going up and up and up because the number of transistors were just increasing. Now, performance then that was great right and in fact, what that meant was that, I'm a hardware person mostly, my background is a lot of hardware. So, I've always felt like software designers, programmers would get lazy. They'd write code and it didn't have to be particularly efficient in terms

of memory or in terms of speed because they knew that pretty soon hardware people would double the number of transistors and fix all their problems for them.

1.1.3 End of Moore's Law

So, that's how it used to be, but that is not happening anymore, because Moore's Law had to slow down. There are several reasons why probably the biggest would be the power consumption and therefore temperature constraints. So, when you pack these transistors onto onto a chip they generate heat. Every time they switch they consume power which generates heat. If you keep increasing the clock rate, then the switch zero to one, zero to one more frequently higher rate and they create more heat and the chip would physically melt. So, if you've ever opened up a machine you see they got fans blowing over the chip. In fact usually, when you open up a box, you see on the box some big heat sink, big gnarly-looking piece of metal that's actually attached to the processor. That's just to distribute heat. So, the wet air blows over at the fan blows over it and distributes the heat so it doesn't melt. So, this is air cooling, right? We are basically at the limits of air cooling. Air cooling can only remove so much heat per unit time. So if you get, if you clock these things much faster with this density of transistors, you're going to melt the thing. So you can't keep increasing the clock rates.

1.2 Parallelism for Increased Performance

So, that's performance limit that's happening with machines, the clock rates are not going up as fast as quickly as they used to go up over time. So, how do you get performance improvement even though you can't just crank up the clock? So, one way to do this is to use parallelism.

Increasing Cores: So, this is typically implemented simple and in a number of ways but you see this as an increasing number of cores on chips, this is one way that you see it, right. So, you got quad core machines, these are common, right. There are four copies of the core on there and you get more. Heck, if you go to a GPU that thing might have a 1,000. 1,000 processor core is all in a inner massive array. So, these cores, the number of cores on the processor increases over time and that helps you because you can perform multiple tasks at the same time, potentially, not always but sometimes. If you've got four cores, you can do four things at once. That can improve your speed, you can get things done faster. Now it doesn't necessarily improve your latency, but your throughput will improve potentially.

Problems with Implementing Parallelism: So, difficulties with implementing parallelism, there are many. But programming wise there are difficulties.

Tasks. So, for instance, when did the tasks start and when do they stop? A programmer has to decide this. For tasks, these tasks are not completely independent.

Sharing of Data. So, what happens when one task needs to get data that's generated by another task, right? How does this data transfer occur?

Memory Conflicts. Also, if you've got multiple tasks running at the same time, do these tests conflict in memory, right? They should not. You don't want one task to write to it's variable A and that overrides variable B in another task.

So, these are all problems that happen when you have concurrent execution going on. Because even if you got multiple cores you got to worry about the memories, are they sharing memories?, do they have separate memories? This is all features in the hardware but the programmer has to often be aware of these things and it's hard. So, writing this type of code, code that can execute in parallel can be difficult.

Concurrency Enables with Parallelism So, in comes concurrent programming. *Concurrency is the management of multiple tasks at the same time.* So, when I say at the same time, they might not actually be executing at the same time. Maybe they're executing on a single core processor. So, they're not actually executing at the same time, but they are alive at the same time. So, they could be executing at the same time if you had the resource, but they need to be going on at the same time. So, maybe one is paused while the other ones running but they are all alive at the same time and need to be handled at least from the user's perspective at the same time. So, this is key for large systems. There are big systems have many things, many pieces going on and they're not all executing sequentially. You want them you want to be able to consider 20 things at one time. Now, maybe they're not actually executing at the same time but you would like to have the possibility of executing them in parallel if at all possible, just for speed. So, concurrent programming it enables parallelism.

Map Tasks to Parallel Resources. So, if you can write program code, write code so that all these tasks can be alive, multiple tasks can be alive and communicating the same time, then if you have the resources, the parallel resources multiple cores, multiple memory stuff like this then you can map them onto those parallel resources and get parallelism.

Partitioning of Code. So, you can't just take a regular piece of code and say okay I'm going to run it on five cores, that won't work. The programmer has to decide how to partition this code. I want this running on one core, this on another, I want this data here this data there and so on. So, that's what concurrent programming is about. The program is making these decisions that allow things to run in parallel. If parallel if the hardware exists.

1.2.1 Parts of Concurrent Programming

So, concurrent program includes several things, we'll go into more depth and layer in the concentration, but specialization rather.

Task Management: But management of task execution, so when our test starts and stops, how do two tests communicate, send data back and forth, share memory if they share memory and how did they synchronize?

Synchronization: So, there are times where one task has to do something for the next task can start. So, there are times where two tasks can't be executed completely in parallel. There has to be some sequential behavior. This test can't start until this task ends and so on. So, that's *synchronization* and you have to be able to manage that inside your programming language. The programming basically have to say, express inside the code where synchronization needs to occur and where it doesn't. So, that's what concurrent programming is and it is important if you want to be able to exploit parallelism when it exists.

1.3 Concurrency in Go

So, concurrency in Go. So, basically the thing about Go, is that Go has a lot of concurrency primitives built-in to the language and implemented efficiently.

Go Routines: So, Go routines, each one of these Go routines represents a concurrent tasks, basically a thread.

Channels: Channels are used for concurrent for communication between concurrent tasks.

Select: Select is used to enable synchronization. These are just the high level basic keywords that you can use. But we'll talk more about these later on in the specialization.

1.4 Summary

But concurrency, having concurrency built into the language and have an efficient implementation is advantageous if you're doing concurrent programming which more and more, especially with all the cores that exists in processes these days has become more and more important.