CONCURRENCY AND PARALLELISM, BETWEEN PROGRAMMING AND REAL LIFE

Marius BÎTCĂ[1*], Darinela ANDRONOVICI[1], Dumitraș MĂMĂLIGĂ[1]

[1]*Technical University of Moldova, Faculty of Computers, Informatics and Microelectronics, Department of Software Engeneering and Automatics, group FAF-191, Chisinau, Moldova*

*Corespondent author: Bîtcă Marius, bitca.marius@isa.utm.md

Abstract. *Undergraduate or novice programmers are often challenged by higher-level and abstract concepts in programming courses. Compared to constructing a sequential program, parallel and concurrent programming requires a different and more complex mental model of control flow. Now that multi-core processors have become the norm for computers and mobile devices, the responsibility of developing software to take advantage of this extra computing power now rests with the modern software developer.*

Keywords: *performance, programming, threads, sequential program, computer architecture*.

Introduction

The aim of this article is to help the reader to understand what parallelism and concurrency are, by bringing not only definitions and explanations, but also examples from real life, as it will be clearer to understand. There are a lot of explanations, but only few of them can give you a good perception about them, the rest are making you feel confused and then you give up on understanding these two terms. You do not even know that you see concurrency and parallelism not only when you are programming, but everywhere and every time.

Real life implementation

Imagine a person is working in a library and a new bunch of books arrived. His task is to select the right ones, by author, and put them on the shelves. The way he will accomplish this task is by following the right steps. From all the books, he will pick those written by the same author. After bringing them to the corresponding place, he will arrange them on the shelves.

In order to make this process more efficient he can implement the parallelism technique, by using two workers and make them work simultaneously. In this way he will reduce twice the amount of time. Of course, if he wants to make this job more efficient, he can use more workers. An important thing to know about parallelism is that sometimes you cannot obtain the expected increase in performance, because you can reach a bottleneck, this happens when a resource (the books) is busy and the second worker is unable to select the needed books, that is why you can lose that same amount of time as if you work with one worker.

Now if you want to optimize even better, you can use the concurrent approach.

So before jumping into this subject by defining what concurrency is, and since it is very easy to confuse concurrent with parallel, we have to try to make the difference clear from the beginning:

- *Parallelism is about **doing** a lot of things at the same time.*
- *Concurrency is about **dealing** with a lot of things at the same time.*

Parallelism

Parallelism means executing multiple tasks on multiple hardware (cores, machines etc.), that is why the tasks are running parallel and they are being executed as fast as possible. A parallel computer is a computer or a system that uses elements of simultaneous processing in a cooperative

manure for solving a computation problem. Parallel processing includes technics and technologies that make the calculation in parallel to be possible.

The goal of parallelism is performance, because we just want to complete the task as fast as possible, therefore the time spent is reduced twice or more and the amount of work per individual is divided. In Figure 1 is an example from real life, where you can see the process of transporting the books to the fireplace by two gophers at the same time.

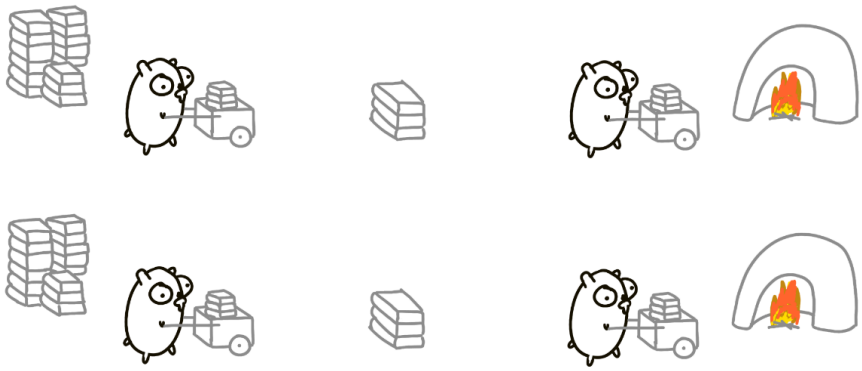

Figure. 1. Example of parallelism

The development of parallel computing has attracted important progress in the parallel algorithms field, whose main feature is the ability to execute several computation operations simultaneously. Initially, parallel computers could not be encountered other than in research laboratories. These systems were used exclusively for scientific applications that required intensive calculations.

Examples of this would be parallel programs intended for numerical simulation of complex systems. However, today the development of parallel computers is mainly imposed by commercial applications that are capable to handle and process large collections of data. Some of the areas where these applications have found a wide applicability are: graphics and virtual reality, parallel databases, expert diagnostic systems and decision support.

It can be stated without doubt that the directions of development of these two categories of applications are approaching one common denominator since commercial applications companies tend to make more and more complex calculation. One of the most common techniques of parallelization is the division of a problem in several subproblems and solving them in parallel with the help of processes and threads.

Concurrency

Concurrency is not a new idea. Much of the theoretical groundwork for concurrent programming was laid in the 1960's, and Algol 68 includes concurrent programming features. Widespread interest in concurrency is a relatively recent phenomenon however; it stems in part from the availability of low-cost multiprocessors and in part from the proliferation of graphical, multimedia, and web-based applications, all of which are naturally represented by concurrent threads of control

You have probably written multiple single-threaded programs before. A common pattern in programming is having multiple functions that perform a specific task, but they do not get called until a previous part of the program gets data ready for the next function.

This is how we will initially set up our first example, a program that mines ore. The functions in this example perform: *finding ore*, *mining ore*, and *smelting ore*. For a single-threaded application, the program would be designed as in Figure 2.

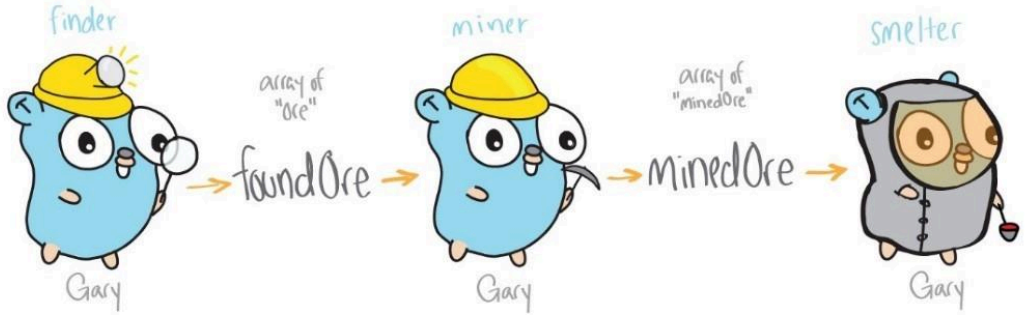

Figure. 2. Program that mines ore

This style of programming has the benefits of being easy to design, but what happens when you want to take advantage of multiple threads and perform functions independent of each other? This is where concurrent programming comes into play.

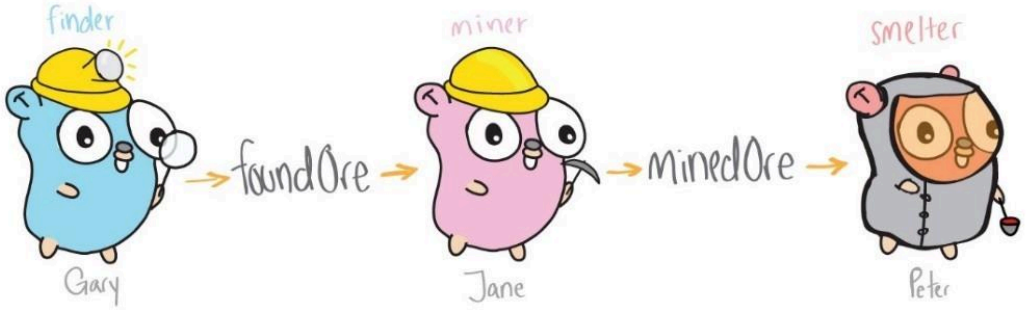

Figure. 3. Program that uses concurrent programming

Mining design, presented in Figure 3, is much more efficient. Now multiple threads (gophers) are working independently; therefore, the whole operation is not all on Gary. There is a gopher finding the ore, one mining the ore, and another smelting the ore – potentially all at the same time.

Problems in concurrent programs

Solving a problem concurrently seems that it will reduce the computational time immensely. However, everything comes at a price. Even though we think that doing many things at once will speed things up, there is a cost due to the communication between threads and to make sure that they won't crash or make wrong outputs. Concurrent programming has to be done with great care and it causes an unavoidable overhead to program.

Some operations in a concurrent program may fail to produce the desired effect if they are performed by two or more processes simultaneously. The code that implements such operations constitutes a critical region or critical section. If one process is in a critical region, all other processes must be excluded until the first process has finished. When constructing any concurrent program, it is essential for software developers to recognize where such mutual exclusion is needed and to control it accordingly.

Usually, an external library will be used when writing concurrent programs. There will be an overhead to load these libraries. Additionally, concurrent programming building blocks like semaphores, mutexes, locks will be used and they will cost an initializing and finalizing time.

Conclusions

The two concepts are related, but different.

- Concurrency and Parallelism refer to computer architectures which focus on how our tasks or computations are performed.
- In a single core environment, concurrency happens with tasks executing over same time period via context switching, when the CPU changes from one task (or process) to another while ensuring that the tasks do not conflict.

- In a multi-core environment, concurrency can be achieved via parallelism in which multiple tasks are executed simultaneously.

As you can see, concurrency is related to how an application handles multiple tasks it works on. An application may process one task at time (sequentially) or work on multiple tasks at the same time (concurrently).

Parallelism on the other hand, is related to how an application handles each individual task. An application may process the task serially from start to end, or split the task up into subtasks which can be completed in parallel.

As you can see, an application can be concurrent, but not parallel. This means that it processes more than one task at the same time, but the thread is only executing on one task at a time. There is no parallel execution of tasks going in parallel threads / CPUs.

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. 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.

Finally, an application can also be both concurrent and parallel, in that it both works on multiple tasks at the same time, 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 analyse and measure before you adopt a concurrent parallel model blindly.

References:

1. FORREY, T., *Learning Go's Concurrency Trough Illustration*[online] [accessed 10.02.2020] Available: https://medium.com/@trevor4e/learning-gos-concurrency-through-illustrations-8c4aff603b3
2. ABHISEK, G., *Concurrency, Parallelism, Threads, Processes, Async and Sync – Related?* [online] [accessed 13.02.2020] Available: https://medium.com/swift-india/concurrency-parallelism-threads-processes-async-and-sync-related-39fd951bc61d
4. JENKOV, J., *Concurrency vs. Parallelism* [online] [accessed 17.02.2020] Available: http://tutorials.jenkov.com/java-concurrency/concurrency-vs-parallelism.html
5. PIKE, R., *Summary of Concurrency Is Not Parallelism, a talk by Rob Pike* [online] [accessed17.02.2020] Available: https://rakhim.org/2019/12/summary-of-concurrency-is-not-parallellism-a-talk-by-rob-pike/
6. NORMAND, E., *Concurrency and Parallelism in the real world* [online] [accessed 19.02.2020] Available: https://lispcast.com/concurrency-vs-parallelism/
7. ALECU, F., *Parallelism Implementation Mechanism* [online] [accessed 21.02.2020] Available: http://revistaie.ase.ro/content/34/alecu.pdf
8. BUSTARD, D., *Concepts of Concurrent Programming*. Curriculum module. Pittsburgh (USA): Carnegie Mellon University. Software Engineering Institute, April 1990.