About an implementation "detail" : a few ideas to speed up the execution and the C++ programming (in Operations Research)

Daniel Porumbel[1]

[1] Conservatoire National des Arts et Métiers, CEDRIC, 75003, Paris, France. daniel.porumbel@cnam.fr

[...] so for the most part the Commodore on the quarter-deck gets his atmosphere at second hand from the sailors on the forecastle. He thinks he breathes it first; but [this is] not so.
Chapter 1 "Looming", Moby-Dick

Introduction

The execution speed of any practical algorithm clearly depends on a constant complexity factor δ closely related to the quality of the implementation. After ten years of talks at this annual (Roadef) congress, I finally realized that all of my talks invariably end with a line like "And finally, I will spare you the implementation details because I prefer to focus on the essentials and on the high level concepts". Despite its merits, this sentence is too simplistic. It is likely to divert a fast reader from the truth. Like a backdoor in software, an unspoken consequence is that the (constant complexity factor associated with) the quality of the implementation has no relevant impact on the total computational cost and can be ignored. This constant complexity factor δ is also invisible when calculating theoretical complexities such as $O(n^2)$, $O(n^2 \log n)$, etc. But imagine a salesman saying that the cost to pay is 1000$ multiplied by a some constant factor $\delta=3$ or $\delta=6$ that has no importance in absolute terms. With all due respect to all rich people for whom 1000$ or 3000$ is more or less the same, I would not let myself be convinced so easily. *I am even very grateful to my destiny for having spared me the emptiness of an existence in such disproportionate luxury.*

Some years ago many used to belittle the constant complexity factor δ because they honestly thought the clock frequency of CPUs will never stop to increase exponentially. But this proved to be simply a mirage partly fueled by a superficial interpretation of the law of Moore. Although the number of transistors in a CPU can indeed double every 18 months, the clock frequency has tended to stagnate since 2004 and some physical limits have been reached. I must confess I have also written papers with no implementation that contain only complexity proofs (e.g., in the theory of sub-modular functions). Still, I really do *not* want to forget about the constant complexity factor δ nor about the programming time needed to implement practical algorithms. If an algorithm is great in theory but too difficult to implement, it will never be successful (think of the ellipsoid method).

Although the theory proposes many tools to determine the computation time, the practice is far more complex and easily escapes the simplifications and axiomatizations of the theory. The number of factors that come into play is incalculable. I usually indicate in my articles a few factors such as: the programming language (C++), the clock frequency, the amount of RAM, the operating system, etc. It is easy to understand I am used to overlook many other aspects, such as the cache hierarchy, the compiler optimizations, the speed of the system bus that manages the access of the CPU to the memory, the hardware quality, and so on -- this list could grow endlessly.

This list is actually so long that it is impossible to enumerate all these practical factors and alll their interactions to determine the value of the constant complexity factor δ. There is only one fact that can ease the task of a programmer working in (optimization) research: we do not necessarily need advanced optimization programming tricks; the functionalities of a rather minimalist language like C are sufficient, at least for writing prototypes. We do not actually need any high-sounding or esoteric programming concept, such as denotational semantics, multiple dispatch, factory methods continuations polymorphism, auto-boxing, complex inheritance relationships, friendly classes, etc – this list may also grow endlessly .

However, I can recall that object-oriented programming was invented by researchers in optimization and Operations Research (OR) who needed more organization and more structure in their code. This led to the Simula language which greatly influenced Stroustrup to design C++. It *cannot* be said that the field of optimization and OR can never have a positive influence on computer science, or that it would be useless to focus on ideas/practices that would ease the burden of OR programmers.

A few programming practices and the age-old advantage of learning by yourself

My talk will also address a few programming practices , that I use to develop C++ applications in optimization and OR (cedric.cnam.fr/~porumbed/CODE_GUIDELINES). Here's a first principle: performance is a priority in optimization and you have to write a code that is as transparent as possible, *i.e.* you have to exactly know what happens when a block of code is executed. I'm *not* going to give advices on how you should write code. The designer of C++ solved this question unequivocally: "C++ is deliberately designed to support a variety of styles rather than a would-be 'one true way' ". In addition, I assume that I have a rather atypical C++ style. In fact, I am living proof that Linus Torvalds was (almost) right when he said "the only way to do good, efficient, and system-level and portable C++ ends up to limit yourself to all the things that are basically available in C."

These practices enabled me to write many optimization algorithms quite rapidly. For example, my second paper proposed this year (at Roadef) relies on a code of about 14000 lines and the programming task took about a third of the total working time (2-3 months). Although I programmed myself the algorithms associated to most of my papers, the coding time was never too important. By programming all algorithms by myself, I realized I benefited from an *age-old advantage concerning all experiences we may have under the sun. Anything that you discover by yourself is unforgettable ; the best way to discover all things is always through direct experience and not through the explanations given by someone else* - in the spirit of the motto of this paper.