Vladislav Nazaruk, Pavel Rusakov
Riga Technical University, Latvia

METHODS FOR ANALYZING PERFORMANCE OF PROCESS SYNCHRONIZATION ALGORITHMS

Abstract
One of the main goals of using concurrent computing is to speed up the computations. This is done by structuring a program in threads in a way to have a possibility of dynamic reallocation of processor time for other threads when one thread is idle. However, the possible speedup and the overall performance depends on many factors, including synchronization algorithms used in a program, a pattern of thread interaction, a pattern of waiting for external events etc. Therefore, before choosing a better process synchronization algorithm for a specific situation, the performance of possible algorithms to be used should be analyzed. Methods for analyzing such performance for different situations are the object of study of the research.
Key words: process synchronization, algorithms, performance

Introduction

Concurrent computing nowadays has become sufficiently widespread. Concurrent computing is defined as a form of computing which supposes that a computational unit (i. e., a computer program) is divided into several computational units (processes, threads) which can be executed not only sequentially, but also simultaneously (or concurrently). The main purpose of such dividing is to be able for the system to execute other units of a program while one (or some) of units of this program is idle (i. e., waiting for some external event). This means that the system is able to dynamically reallocate processor time between threads when some thread becomes idle. Such reallocation allows to decrease the time needed for the program to execute, compared to equivalent non-concurrent program. In addition, on systems with more than one processing unit in them (for example, in a multi-processor system or in systems with a multi-core processor), the presence of several threads allows to execute more than one thread at each time moment — this is also the opportunity to decrease execution time of a program. Therefore the main goal of using concurrent computing is to reduce the execution time of a program or, which is equivalent, to speed up the computations. (Downey 2008: 1–2)

One of the main aspects of concurrent computing is that in order to accomplish a corporate goal for a program, its threads should communicate with each other in a certain way. All communications in a concurrent program are done by means of inter-process (or inter-thread) communication mechanisms. Process synchronization is one of most important types of inter-process communication; its aim is to assure a specific coherence of execution of actions between several threads (or processes).

As process synchronization mechanisms control the execution of threads, these mechanisms, operated by their own logic, pause and resume different threads. Such intermediation of synchronization mechanisms in the execution of other parts of a program obviously impacts the execution speed. As there exist a number of different process synchronization algorithms, and the same concurrent computational task can be implemented in different ways by using different synchronization algorithms, the analysis of the impact of process synchronization mechanisms on the execution speed of concurrent programs is a fairly topical issue; it is in the focus of this paper.

The goal of this paper is to suggest and analyze methods how such impact of process synchronization mechanisms (or, in other words, the performance of synchronization mechanisms) on the execution speed for different situations could be measured and interpreted. By applying these methods in practice, for example, it would be possible to predict some issues concerning the overall performance of a given concurrent program when using different synchronization algorithms; this, in its turn, can help develop guidelines for selecting more appropriate process synchronization algorithms in specific or more general situations.

1. Performance metrics

In order to measure the performance of process synchronization algorithms, it is necessary to introduce some requisite metrics, or measures of some properties, of a concurrent program.

For ease of describing, let all metrics be divided into two classes: observational metrics and analytical metrics. Observational metrics are metrics which are obtained by direct measurement of properties; and analytical metrics are metrics obtained by calculations over observational and/or other analytical metrics.

Let performance metrics also be divided into two classes, depending on their scope: thread-level metrics and system-level metrics. Thread-level metrics are specific to each thread of a concurrent program, and are obtained considering the corresponding thread outside the context of other threads. On the contrary, system-level metrics are specific to a program in general.

Before discussing specific metrics, some general assumptions which will allow formalizing the results, should be formulated. They are the following:

- given a computer program, all metrics are affected only by some input parameters (defined later in this paper) — that is, they are not affected by some random factors;
- the problem is finite (and, therefore, completion time of the program is also finite);
- all threads start their execution simultaneously.

Now let us consider thread-level metrics. In Figure 1, there is schematically shown lifetime of a thread. One can define two principal possible states of a thread: performing effective actions (s_{eff} — all actions a thread performs in isolation) and synchronizing with other threads (s_{syn}). These states usually alternate with each other; and the alternation could occur an unlimited number of times. The s_{syn} state has its two substates: executing instructions needed to provide the synchronization ($s_{\mathrm{syn:ex}}$) and waiting for synchronization to complete ($s_{\mathrm{syn:wait}}$).

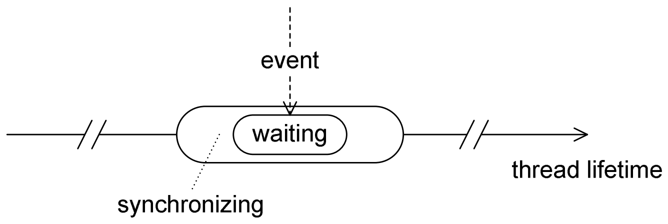

Figure 1. Thread lifetime

Let us assign to each state s_a the total time t_a the thread is in this state. Analyzing the thread-level metrics, it is clear that for each thread effective time t_{eff} is constant; however, $t_{\mathrm{syn:ex}}$ and $t_{\mathrm{syn:wait}}$ are both variable, and for better performance they should be minimized. It is also important to say that the time $t_{\mathrm{syn:ex}}$ is dependent only on the synchronization mechanisms (including their implementation) used in the thread; however $t_{\mathrm{syn:wait}}$ is dependent both on the synchronization mechanisms (*excluding* their implementation) used in the thread, and as well as the behaviour of other threads — which is far less predictable.

Considering system-level observational metrics, there could be defined overall execution time (T_{ex}) — a time interval between the beginning and the end of execution of the entire concurrent program. Other system-level metrics could be defined as follows:

— total effective time (T_{eff}) — the sum of effective time (t_{eff}) for all threads;

— total waiting time ($T_{\mathrm{syn:wait}}$) — the sum of waiting time ($t_{\mathrm{syn:wait}}$) for all threads; etc.

When measuring overall performance of synchronization mechanisms in a concurrent program, the following analytical system-level metrics are fairly significant:

— effectiveness rate $= \frac{T_{\mathrm{eff}}}{T_{\mathrm{ex}}}$: shows how effectively was executed actual work; if a number of processing units (or cores in a processing unit) is n, the best values of this measure converge to $\frac{1}{n}$, however, this is possible only when threads almost do not depend on each other;

— synchronization execution rate $= \frac{T_{\text{syn:ex}}}{T_{\text{eff}}+T_{\text{syn}}}$: shows the rate of operating costs for executing instructions in synchronization algorithms; takes values in interval $[0;1)$ (0 is the best, 1 is the worst); this metric depends only on synchronization algorithms used in separate threads and their implementation;

— synchronization waiting rate $= \frac{T_{\text{syn:wait}}}{T_{\text{eff}}+T_{\text{syn}}}$: numerical interpretation is similar to the previous metric; however, unlike the previous, this metric shows a holistic result which depends not only on summation of independent results of separate threads, but also on a way that different threads interact with each other.

2. Factors that metrics can depend on

When working with metrics, it is important to know nearly all factors (inputs) that metrics can depend on. In this section, the authors of this paper tried to identify such factors.

Firstly, factors on which can depend performance metrics should be split into two parts: program-dependent and environment-dependent. The latter metrics usually imply some significant properties of program execution environment (usually hardware), including:

— a number of processing units (or number of cores in a processing unit),

— type and architecture of a processing unit,

— performance of a processing unit; etc.

Program-dependent factors mostly depend on a structure and synchronization logic of threads and include the following:

— number of threads,

— structure of each thread: how much effective work a thread should do before each act of synchronization,

— pattern of thread interaction,

— pattern of waiting for external events,

— synchronization mechanisms used in each synchronization time; etc.

3. General considerations about approaches to measurement

When there are defined basic performance measures of process synchronization algorithms and main factors which influence the results of these measures, it is needed to define a way how the measurements could be done. Potentially there are two different approaches to such measurements: observation of real systems and simulation (see (Hartmann 2005: 5–7)).

The first approach consists of taking a real computer program (or writing a “synthetic” program with needed properties), adding to its code some time measurement routines which will measure the time intervals t_{eff}, $t_{\mathrm{syn:ex}}$, $t_{\mathrm{syn:wait}}$ and T_{ex}, and running the program (possibly, multiple times to obtain higher measurement precision). This approach has the following characteristics:

- — measurements can be done relatively easy (by inserting in a program a relatively simple measurement code);
- — the precision of measurements will suffer due to the measurement code will interfere with the parts of base program;
- — time needed to obtain results could be reasonably large due to real-time execution; moreover, many execution times would be necessary if there is a need to obtain the measurements for different input parameters;
- — there will be rather hard to generalize the obtained results due to a large number of influencing factors.

The second approach consists of modelling a concurrent system taking as inputs some most important general properties (factors) that could describe the system (however, some amount of work should be done to select from a list of properties those which are most appropriate), and simulating its execution, where measurement results are fixed by the simulating environment. This approach could be characterized by the following facts:

- — to apply this approach, there is needed for a strong mathematical model of a structure and operational logic of thread synchronization (preferably, such model could be based on Petri nets; see (Winskel, Nielsen 1993:80));
- — the results will be obtained mush faster due to non-real-time execution of the model;
- — it would be easier to see the correlation between input information and measures, and for the analysis of the results, mathematical methods could be used (results would be more statistic-oriented) — this is mostly due to the minimization of influencing factors.

Conclusions

The gain of applying concurrent computing is a possible speedup of the overall performance of a program. However, this gain depends not only on the environment of the program, but also on the program itself, including its separation into threads, and types and patterns of their interaction (and considerably synchronization). There exist several uncomplicated metrics which can be used to get quantitative values of the impact of synchronization mechanisms on the execution speed. However, it could be an issue the way how to obtain the measurement results. As two main alternatives, either real system observation or simulation can be used. The former method is easier, but taking more time and much harder in context of generalizing the results; the latter method is more complicated, but faster and easier for generalization.

Possible directions for further work are the following. Firstly, it is rather necessary to maximally simplify the input data set (i. e., factors that metrics can depend on) in order to easier understand the dependencies between these input data and the measures. Secondly, it will be useful to formalize the description of such input data set and to define requirements for software which can simulate a specific interaction of threads and measure needed metrics.

The further work may also include:

— implementing test software that can model the execution of concurrent programs given a specific input data set, and give the measurement results;

— analyzing the performance of widely-used process synchronization algorithms in different use-cases.

Acknowledgements

This work has been supported by the European Social Fund within the project "Support for the implementation of doctoral studies at Riga Technical University".

Bibliography

1. Downey A. B. (2008) *The Little Book of Semaphores.* 2nd ed. http://greenteapress.com/semaphores/downey08semaphores.pdf
2. Hartmann S. (2005) *The World as a Process: Simulations in the Natural and Social Sciences.* http://philsci-archive.pitt.edu/2412/1/Simulations.pdf
3. Winskel G., Nielsen M. (1993) *Models for Concurrency.* http://www.daimi.au.dk/PB/463/PB-463.pdf