Olympiads in Informatics, 2025, Vol. 19, 27–33

DOI: 10.15388/ioi.2025.02

Virtual Time Measurement in Programming Contests

Paweł DIETRICH[1], Bartosz KOSTKA[2]
[1]*Google Poland,*
[2]*Google Canada*
e-mail: p.dietrich@fri.edu.pl, kostka@oij.edu.pl

Abstract: We propose a novel method for measuring time in programming competitions. This method utilizes the instruction count, using existing hardware capabilities, to ensure a fair and deterministic evaluation. By basing the measurement on the count of executed instructions, the method becomes less dependent on the underlying machine's specifications. Furthermore, it facilitates scalability by enabling parallel program execution on multiple threads. We also discuss the adoption of this method in various Polish competitions.

Keywords: virtualized time measurement, hardware performance counters, deterministic program evaluation, programming contest judging systems.

1. Introduction

The goal of this paper is to share a novel approach to measure execution time of participants' submissions developed for Polish Olympiad in Informatics (POI). The authors also share their experience with using this approach in a production environment.

Traditionally, competitive programming problems challenge participants to write a program that performs computations and that execution is restricted by certain time and memory limits. The memory footprint of the program is straightforward to measure, as the approach to handling memory has not changed in years. On the other hand, there are multiple CPU architectures, and within a single architecture, there are different revisions of processors. Participant submissions can run faster or slower depending on the clock speed, cache sizes, CPU die layout, memory controller latency, and many other factors.

Organizers wanting to provide consistent judging verdicts using standard time measurement techniques need to maintain a uniform fleet of judging machines and carefully control their software. To achieve time measurement accuracy within an error margin of a few percent, only one program can be judged on each judging machine. This approach also disallows the use of shared hardware (e.g., cloud machines) due to "noisy neighbor" problems. The requirement to use physical machines is usually costly, and this

approach often wastes significant resources, since today's machines have multiple CPU cores. However, to maintain the time measurement accuracy, only one submission can be judged at a time.

2. Implementations

There were multiple iterations of the mechanism that we use today for time measurement in Polish competitions. The first version of the tool called oitimetool Acedański (2009) used Intel's library PinLuk *et al.* (2005). The current version used in POI is called sio2jailDubiel (2018) was based on perf API.

The first tool was limited to running only on Intel CPUs, and since there are other popular vendors in the CPU space, we decided to switch to sio2jail exclusively and abandon oitimetool. This way, we depend on an open-source software component (the Linux kernel with perf enabled), rather than on a proprietary CPU interface. Both solutions were intentionally implemented to calculate the same time for the same programs. The idea was to make the switch a small technical nuance.

2.1. *Using a Linux perf*

Modern computer systems are equipped with various ways of collecting and analyzing performance data. One of the tools that uses these build-in systems is perf – a powerful performance analysis tool available on Linux systems. perf was introduced as a tool for using the Performance Monitoring Units (PMUs) hardware in modern processors to collect data on hardware events such as instruction counting or cache monitoring.

The perf tool provides a very simple command line interface. For example:

```
$ perf stat -B dd if=/dev/zero of=/dev/null count=1000000

1000000+0 records in
1000000+0 records out
512000000 bytes (512 MB) copied, 0.956217 s, 535 MB/s

 Performance counter stats for 'dd if=/dev/zero of=/dev/null count=1000000':

          5,099 cache-misses             #      0.005 M/sec (scaled from 66.58%)
        235,384 cache-references         #      0.246 M/sec (scaled from 66.56%)
      9,281,660 branch-misses            #      3.858 %     (scaled from 33.50%)
    240,609,766 branches                 #    251.559 M/sec (scaled from 33.66%)
  1,403,561,257 instructions             #      0.679 IPC   (scaled from 50.23%)
  2,066,201,729 cycles                   #   2160.227 M/sec (scaled from 66.67%)
            217 page-faults              #      0.000 M/sec
              3 CPU-migrations           #      0.000 M/sec
             83 context-switches         #      0.000 M/sec
    956.474238 task-clock-msecs          #      0.999 CPUs

      0.957617512  seconds time elapsed
```

There are two main metrics, which can be used to model execution time of the program itself. They are `cycles` and `instructions`. Essentially different instructions can take different amount of cycles. From our experiments the ratio of cycles per particular instruction can vary from CPU to CPU, even within a particular vendor. For more accurate results for a particular hardware this can be a good metric. However our original goal was to be more hardware agnostic, that is why we choose the instructions counter, which counts the literal instructions written in a program's binary file as they are executed.

In sio2jail, we use the perf interface using its C++ API (`linux/perf_event.h`). We set up performance monitoring using `perf_event_open`[1] system call. We set it up to count only instructions from user namespace and only after the execve call (the call that executes the program to be judged) is executed. Additionally, we want to implement the instruction limit. We use the `sample_period` and `wakeup_events` options, which allow us to receive a notification when the counter values exceed certain values, to do so.

2.2. *sio2jail Usage*

The implementation of sio2jail is available in the GitHub repository SIO2Project (2018). The implementation offers process sandboxing using seccomp The Linux Kernel Documentation (2025) and instruction counting using above mentiond perf, memory limit verification and more. There is also a test program available, that was running exactly 1 second on a reference machine and was used to properly scale instructions to seconds ratio.

Unfortunately we did not provide any default options for the sio2jail binary. The users are responsible for setting all the options by themselves. We have released an extra script called `oiejq`, which sets the sio2jail command line options to print `time(1)` like output, which is handy for the participants to use.

3. Pros and Cons

Using this special virtual environment for time measurement can be beneficial for programming competitions in a number of ways.

First, the described method, leveraging hardware event counters and potentially running in a virtualized environment with namespaces, eliminates the dependency on the specific physical configuration of the machine evaluating the program. This means the same program, when run on different machines with varying factors like clock speed, cache size, or even the number of cores (assuming they don't impact the algo-

[1] `https://man7.org/linux/man-pages/man2/perf_event_open.2.html`

rithm's execution path), will result in highly consistent instruction counts. This has several advantages:

- Flexibility in judging machines. The judging machines do not need to have identical configurations, which is often impractical and costly to achieve, especially for larger competitions. This also allows for the judging machines to differ from the contestants' machines. As long as the virtual environment with hardware event counters is implemented, the measurement conditions remain consistent.
- If the environment is published, the contestants also have an ability to run their programs in the exact same environment. This is extremely helpful for situations where we cannot guarantee that each contestant has access to the exactly same machine (for instance in online competitions where contestants participate from their homes, using their personal machines). The contestants do not have to experiment how much smaller/faster are the judging machines compared to the machine they use for the competition, as they can run their programs in the exact same environment.
- Problem setters can set the time limits for their problems much easier. They can now execute their model solutions on their own machines and confidently set appropriate time limits for the problems based on the measured execution time during the development stage. This eliminates the need to worry about hardware discrepancies between their machines and the judging environment.
- A cornerstone of the proposed method is its guarantee of deterministic and repeatable execution. This ensures that the measured execution time for a given solution remains invariant across multiple runs. This characteristic proves particularly advantageous in scenarios where a solution requires re-evaluation. Under the conventional time measurement approach, such re-judgement can lead to discrepancies in the verdict, especially when the solution is fairly close to the time limit. Traditionally, this necessitates executing the solution multiple times and verifying if any of those runs fall within the allotted time. The virtual time measurement method effectively eliminates this issue. Furthermore, the inherent consistency of virtual time measurement facilitates the seamless portability of problems across different platforms, provided they share the same virtual environment. This eliminates the need for time limit re-calibration for each platform, As we use distinct platforms for competitions and public training purposes, this feature streamlines problem reuse across these platforms.
- Simplified maintenance and upgrades. If machines need to be replaced in the judging environment, the consistency of time measurement ensures that time limits for past problems don't need to be recalculated or reevaluated.

Furthermore, we want to highlight that this virtual environment allows for a high degree of isolation for program execution. This isolation ensures that the measured execution time remains independent of extraneous processes running concurrently on the judging machine. Consequently, the time measurement becomes a more accurate reflection of the program's intrinsic performance, as factors like CPU or memory utilization

by other processes are effectively eliminated from the equation. This isolation offers a significant advantage for both contestant and judging machines. For judging machines, this isolation permits the parallel execution and evaluation of the programs. By leveraging multi-core processors, the judging process can achieve significant efficiency gains. Additionally, the isolation allows us to utilize virtualization techniques without concerns regarding the impact of other processes running within the same virtual machine. This allows for greater flexibility in resource allocation and management within the judging environment.

While the proposed method offers substantial advantages, especially for the contest organizers, we have to acknowledge that it represents an abstraction of real-world time measurement. While the discrepancies between the standard method and the virtual processor approach are minimal, we recognize that transitioning to a new time measurement method necessitates a comprehensive preparation and education effort. Our experience, encompassing over a decade of utilizing virtual time measurement, instills confidence that this method will have a negligible impact on the competition experience for the vast majority of participants, particularly for new contestants. This is contingent upon providing them with appropriate tools to gauge execution time within the virtual environment.

The main differences to traditional approach, which can be viewed as cons are as follows.

- Uniform memory access cost. Cache size does not affect the program performance as measured by the number of instructions. It means that programs do not get any penalty due to cache misses and the whole memory basically has a uniform access cost, which is far from the reality.
- A separate realtime limit is still needed. This is because a program, can still hang on a syscall or even just sleep. In such cases no instructions are being executed and a instruction limit might not be reached at all. However in competitive programming usage of sleep and complex syscall instructions is very rare.

4. Adoption in Polish Olympiad in Informatics

Recognizing the significance and potential ramifications of adopting a new time measurement system, we prioritized a well-defined implementation process. This ensured thorough preparation and understanding for both the scientific committee and the contestants.

During the initial phase (2008), all first-stage submissions for the Polish Olympiad in Informatics (POI) were re-evaluated using the oitimetool after the competition concluded. This analysis compared results generated by oitimetool with those obtained through conventional time measurements. The results exhibited a high degree of statistical similarity. Notably, correlation coefficients remained consistently above 0.998. Detailed findings are available in Acedański (2009).

The oitimetool system's first official integration into the competition occurred during the final stage of the POI in 2011. This stage was chosen strategically due to the smaller number of advanced participants (under 100). During this contest, contestant submissions were judged in two distinct environments:

- The traditional environment measuring actual execution time.
- A virtual environment utilizing the Pin library for instruction counting.

The final score for each submission was determined by the higher value obtained from these two executions.

Following the successful pilot run, oitimetool was adopted as the sole judging environment for the first time during the subsequent school year (2011/2012). All three stages of the POI utilized oitimetool, resulting in [number] submissions being judged.

Over the following years, advancements in technology necessitated adaptation. This included the introduction of new hardware-based instruction counting and mechanisms for program privilege isolation and restriction. To address these changes, a modernized judging environment named sio2jail was implemented in 2018. Built upon state-of-the-art technologies of the era, sio2jail addressed some of the limitations present in oitimetool.

The virtual environment was not incorporated into other Polish programming competitions. Notably, both the Polish Collegiate Programming Competition (Akademickie Mistrzostwa Polski w Programowaniu Zespołowym) and the country's largest open competition, Algorithmic Engagements (Potyczki Algorytmiczne), opted out. This decision was based on the belief that both contestants (students and professionals) and problem-setters in these competitions possess sufficient experience with traditional real-time measurement approaches, and therefore, the virtual environment wouldn't offer them significant benefits.

We believe that this implementation allowed us to use full potential of multithreaded judging machines, minimizing impact of the hardware to the scoring and running costs. On the downsides, it slightly changed the characteristics of a CPU and treated memory accesses as uniform.

References

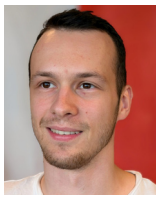
Acedański, S. (2009). *Wykorzystanie sprzętowych liczników zdarzeń do oceny wydajności algorytmów* (Unpublished master's thesis). University of Warsaw.

Dubiel, M. W. T. D. P. J. K. W. (2018). *Sio2jail, narzędzie do nadzorowania wykonania programów zgłaszanych w ramach konkursów algorytmicznych.* (University of Warsaw, 2018)

Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., ... Hazel-wood, K. (2005). Pin: building customized program analysis tools with dynamic instrumentation. *Acm Sigplan Notices*, 40(6), 190–200.

SIO2Project. (2018). *A tool for supervising execution of programs submitted in algorithmic competitions.* Retrieved from `https://github.com/sio2project/sio2jail`

The Linux Kernel Documentation. (2025). *seccomp: secure computing.* Retrieved from `https://www.kernel.org/doc/html/latest/userspace-api/seccomp_filter.html`



P. Dietrich is an engineer at Google Poland working on platforms and infrastructure. He is serving as a member of Polish Olympiad in Informatics Committee focusing on technical details. Paweł is leading a technical team that organizes most competitive programming events in Poland. This team was responsible for running contests with time measurement approach described in this document.

B. Kostka is a software developer at Google Canada. He is actively involved in the competitive programming community, serving as a member of the IOI Scientific Committee, Vice President of the Polish Junior Olympiad in Informatics, and a member of the Universal Cup organising committee. Previously, he worked with many talented students who won multiple IOI medals and also organised and set problems for Google Kick Start.