Review report on the PhD thesis "Performance in Software Development Cycle: Regression Benchmarking" by Tomas Kalibera at Charles University in Prague

Dr. L. Eeckhout – Ghent University, Belgium

The subject of this PhD thesis is regression benchmarking. The goal of regression benchmarking is to complement regression testing by enabling the automatic detection of performance changes during the software development cycle. The idea is to benchmark the software that is being developed on a daily basis so that performance 'errors' (unexpected performance degradations) are identified as early as possible in the design cycle.

The thesis claims to make three contributions to the area of regression benchmarking. First, the thesis proposes a method based on statistics to identify performance changes in the presence of non-determinism, i.e., the regression benchmarking system should not yield false alarms because of random fluctuations due to non-determinism. Second, the thesis proposes a fully automated regression benchmarking environment for a large open-source project, namely Mono. This environment consists of various software tools for downloading the latest version of the software, compile it, evaluate it using a number of benchmarks, collect and analyze the results and publish the results online. Third, the thesis also presents a generic environment that could serve as an aid for building regression benchmarking within other software development projects.

I believe this thesis makes two important achievements. The first major achievement is the building of a solid theoretical underpinning for regression benchmarking that allows for making statistically valid conclusions in the presence of non-determinism. Non-determinism is a complex issue to deal with in performance analysis, and this thesis proposes an approach for addressing it. The second major achievement in this thesis is to show that regression benchmarking can be used in practice during the software development cycle. The regression benchmarking infrastructure built for Mono serves as a proof of concept. As such, this thesis is an interesting mixture of fundamental research on regression benchmarking along with a great deal of hard engineering work. I believe that this is a major contribution to the state-of-the-art in regression benchmarking. As such, I recommend that the PhD degree be granted.

This is the set of general comments/questions that I have that could be posed during the PhD defense.

- How fast does regression benchmarking need to be? Or, how slow may it be without becoming impractical? Should it be done in the order of minutes, hours, overnight? Can regression benchmarking be applied while developers are still working on the code? The latter could be important for large software projects that are being developed by various development groups located on different locations around the world.
- Is it important to apply regression benchmarking on multiple platforms? Or is applying regression benchmarking on a single platform enough? My intuition says it should be done on multiple platforms. In some cases performance improvements or degradations can be a result of particular interactions between the application, the compiler and the processor system. Performance improvements or degradations as a result of such interactions may manifest themselves on a particular platform, but may disappear on other platforms. Can you comment on that?
- A somewhat related question. What do you consider a performance 'error'? For example, by changing the layout of a data structure your application may at the sudden run 20% slower or faster. Is that considered a performance error? Note this may be a result of a particular cache configuration on the machine on which the regression benchmarking is done. On another platform, the performance degradation or improvement may not appear.

- When evaluating a regression benchmarking environment, one needs to know when a false alarm occurs. How do you detect that a performance change is a false alarm? Does this require manual inspection of the code? How did you handle the detection of false alarms during your thesis work?
- The thesis focuses on regression benchmarking for long-running applications. This means that a warmup period needs to be determined after which the regression benchmarking starts its measurements. However, in some cases startup time is also very important. For example, startup time for a virtual machine is an important issue. I guess regression benchmarking can also be applied to evaluating application startup time.
- Can you give a more detailed explanation on what the issues are with warmup. Where does warmup come from? Is it due to warming cache state, TLB state, memory state, etc? What's the time period we're talking about here. Cache and TLB warming is done in the order of millions of instructions (a few hundred millions at most). Warming memory probably takes longer especially for commercial applications such as databases with very large working set sizes. Are we talking about seconds or minutes or tens of minutes?
- When using regression benchmarking in practice, I believe another issue is to evolve the benchmarking itself. The reason is that as new features are being added to the software under development, these features need to be evaluated in terms of performance. As a result, there is a need to constantly innovate the benchmarks that are run during regression benchmarking. Can you comment on that? Along the same line, do you believe that the benchmarks that you use are representative for real applications for identifying performance errors in the Mono environment? If the benchmarks do not stress the potential performance errors, the performance errors may not become visible during regression benchmarking.
- It would be instructive if you would have a case study showing that regression benchmarking effectively points where in the source code you have observed a performance error and how you can actually solve that performance error.
- Can you elaborate on the various sources of non-determinism in today's computer systems? The thesis only mentions two sources of non-determinism, namely memory allocation and compilation. I believe other sources are interrupts, system calls, microarchitecture effects (different memory allocation may affect TLB behavior) and non-constant memory and network latency effects (see also the paper by Alameldeen and Wood as discussed later). Another form of non-determinism I believe is temperature regulation in most of today's microprocessors. Temperature regulation may scale down the clock frequency (and voltage level) when the chip is heating up. Obviously, this adds to the total execution time of the application.
- Can you describe the relation between your work and the work being pointed out below?
  - There exists a large body of work on using hardware performance counters for better understanding program behavior. See for example – just to name a few – the VTune tool developed by Intel, the vertical profiling work done by Hauswirth, Sweeney *et al.* (published at VM'04, OOPSLA'04 and OOPSLA'05), approaches for linking hardware performance counter measurements to the source code (see the work done by Georges *et al.* at OOPSLA'04), etc.
  - There exists a large body of work on visualizing program behavior as well. Several big computer companies such as IBM, HP and others have various tools for gaining insight through performance visualization. This is especially valuable for analyzing large scale systems.
  - Non-determinism in the context of architectural simulation was discussed by Alaa R. Alameldeen and David A. Wood in their HPCA'03 paper. That paper also discusses how non-determinism can be handled using statistics. Can you provide a detailed comparison between your work and Alameldeen and Wood's work?
- At the top of page 11, performance profiling is discussed. Do you refer to profiling through instrumentation or through hardware performance counters? Profiling using

hardware performance counters does not incur overhead in contrast to what is being described in this paragraph. The author seems to focus on performance profiling through instrumentation.

- Page 14 states that the regression tests could be written by the source code developers. I think that testers are required for evaluating the code that are not part of the source code development team. The tester should be unaware of the source code in order to set up unbiased tests.
- Page 21 states that "the compiler uses intentional randomization". Can you elaborate a little more on that? What is it? Why is that?
- Page 35 explains what a complex benchmark is supposed to do. The example that is being used there is a simplified version of TPC-W in that "it collects the duration of individual operations rather than a single value of throughout". I would think of the individual operations being the simple benchmarks, and the combinations of all these simple benchmarks to form a complex benchmark. Is that correct? If yes, what's the difference then between a simple benchmark and a complex benchmark?
- Pages 23 and 24 state that "non-deterministically failing benchmarks should be restarted". A benchmark failing now and then, shouldn't this be considered as an error in functionality? Isn't this a case that should be covered by regression testing rather than regression benchmarking?

[redacted]

Lieven Eeckhout
July 12, 2006
Ghent University, Belgium