Exploring Adaptive Program Behavior

Bonnichsen, Lars Frydendal; Probst, Christian W.

Publication date:
2014

Document Version
Publisher's PDF, also known as Version of record

Exploring adaptive program behavior

Lars Bonnichsen, Christian W. Probst

1 Abstract

Modern computer systems are increasingly complex, with ever changing bottlenecks. This makes it difficult to ensure consistent performance when porting software, or even running it. Adaptivity, *ie*, switching between program variations, and dynamic recompilation have been suggested as solutions. Both solutions come at a cost; adaptivity issues a runtime overhead and requires more design effort, while dynamic recompilation takes time to perform. In this project, we plan to investigate the possibilities, limitations, and benefits of these techniques. This abstract covers our thoughts on how adaptivity and dynamic recompilation can be integrated and evaluated.

2 Introduction

Most problems can be solved in many ways. The solution variants make different tradeoffs, such as trading off space for time. For instance, there are many sorting algorithms offering different tradeoffs, each with numerous implementations. Which variant is best depends on the hardware it runs on, the data being processed, the system load, the power profile, *etc.* Traditionally applications are deployed once and run many times, making it unlikely to deploy the best solution.

Deciding which solution to use before knowing the exact conditions it operates under is at best an art and at worst guesswork. Adapting program behavior through adaptivity or dynamic recompilation, aims to make consistently good decisions. The system decides which solution is used based on observations of the system. Recompilation changes the implementation at runtime by recompiling the program. Adaptivity changes the behavior of programs by deciding which pre-compiled solution to use at runtime. Use of adaptivity requires developing multiple solutions, while recompilation generates new solutions. The new solutions are unlikely to differ as significantly.

In the past 15 years the advantages of adaptive program behavior have been thoroughly studied. Recompilation and adaptivity have been used to adapt programs based on; *e.g.* compiler flags, branch alignment, loop unrolling, prefetching, parallelization, and algorithmic choices. In many cases adapting program behavior can dramatically improve performance and consistency.

Adaptive program behavior is not without its faults. Finding new solutions is costly, and allowing for such changes may harm optimization opportunities. Worse, most adaptive solutions are domain specific and difficult to integrate and debug. Often the costs may outweigh the benefits of briefly using a better solution.

Deciding if adaptive program behavior is beneficial for a new problem, before trying it out, is at best an art and at worst guesswork. We aim to investigate the costs and benefits of adaptive program behavior, so that it can be applied with consistent results. To do this, we will compare integrated and stand alone adaptivity and dynamic recompilation against state of the art optimization techniques on real life applications.

At the core, we want to discover under which conditions:

1. adaptivity is able to achieve good and consistent performance,
2. dynamic recompilation is able to achieve good and consistent performance?
3. traditional compilation techniques is able to achive good and consistent performance?

3 Adaptive program architecture

We propose a simple feedback loop, where applications are iteratively refined. The overall structure is outlined in Figure 1. Adaptive behavior is specified through code annotations, by providing multiple solutions to problems. The compiler uses the annotations to generate adaptive code, and to refine its compilation procedure. While executing on a system, the program adapts to external events. When the system state changes sufficiently the program is recompiled based on observed system state.

The compiler keeps adaptivity in check. When enough information is available, the compiler decides which solution to use, avoiding the cost of adaptivity. When opportunities for adaptivity are detected, the compiler may reintroduce or generate variants of solutions.

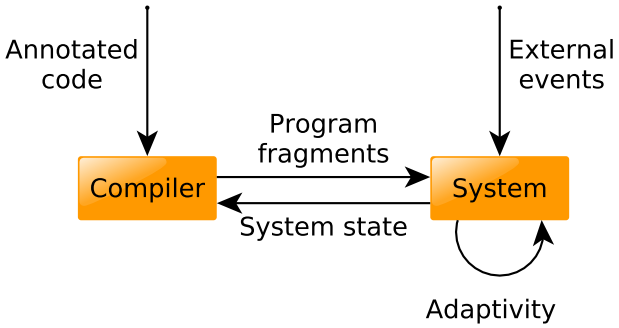

Figure 1: The overall structure

The system state is monitored fully to detect opportunities for adaptivity. At runtime we will monitor the system state through performance counters, OS events, and executed code patterns. The system state guides adaptivity and significant changes trigger recompilation. When compiling, the system state is correlated with the compilers intermediate representation, to evaluate the strength of solution variants.

We will investigate where and why recompilation and adaptivity are beneficial on real life applications. Any benefit from adaptive behavior will be scrutinized, to answer the following:

- Can similar benefits be achieved without adaptive behavior?
- Do the benefits outweight the disadvantages?
- Is it portable to other systems?

4 Conclusion

Adaptive program behavior has enormous potential, but it is difficult to apply well. We aim to reduce the costs of adaptive program behavior, by studying when and how it can be applied in diverse scenarios. In the research project we will implement and integrate adaptivity and dynamic recompilation using feedback loops based on the observed system state. The various forms of adaptive program behavior will be compared against well studied compilers on real life applications.