Scalable Analyses via Machine Learning: Predicting Memory Dependencies Precisely

Lars Gesellensetter

Software Engineering for Embedded Systems,
Institute for Software Engineering and Theoretical Computer Science,
Technical University of Berlin, FR 5-6,
Franklinstr. 28/29, 10587 Berlin, Germany
`lgeselle@cs.tu-berlin.de`
`http://pes.cs.tu-berlin.de/`

Abstract. Program analysis tackles the problem of predicting the behavior or certain properties of the considered program code. The challenge lies in determining the dynamic run-time behavior statically at compile time. While in rare cases it is possible to determine exact dynamic properties already statically, in many cases, e.g., in analyzing memory dependencies, one can only find imprecise information. To overcome this problem, we look at the dynamic run-time behavior and use Machine Learning (ML) techniques to learn the relationship between static program features and dynamic run-time behavior. ML yields highly scalable predictors, which are safely applicable when erroneous predictions merely have an impact on program optimality but not on correctness. Once trained, these predictors can be used in a static compiler.
In this extended abstract, I present our approach to mitigate the impact of the memory gap using ML techniques and speculative optimizations. The memory gap denotes the fact that over the last decade, computer performance is increasingly dominated by memory speed, which did not manage to keep pace with the ever increasing CPU rates. We consider novel speculative optimization techniques of memory accesses to reduce their effective latency. We trained predictors to learn the memory dependencies of a given pair of accesses, and use the result in our optimization to decide about the profitability of a given optimization step.

Keywords. Program Analysis, Memory Dependencies, Speculative Optimization, Machine Learning

1 Overview

Program Analysis tackles the problem of predicting the behavior or certain properties of programs. This is inherently difficult for static analyses, since they have to determine the dynamic run-time behavior statically at compile time. While in special cases, static analyses can predict the dynamic behavior exactly, most of the time precision has to be sacrificed for scalability, in order to process real-life software. For analyses that have to be safe (i.e., must not err), this means that the result is an overapproximation of the actual behavior.

For some applications, this may suffice. For the analysis of memory dependencies (also known as *alias analysis*), however, the imprecision of the analysis has a significant impact on optimization potential and therefore on run-time performance. Hence, very precise analyses are vital for good performance results. This is especially important in presence of the *Memory Gap.*

The *Memory Gap*, also known as *Memory Wall*, describes the fact, that, while both CPU and memory speed have been growing exponentially over the past decades, the speed of memory did not manage to keep pace with that of CPUs. As a consequence, the CPU may have to wait several hundred cycles until a value has been fetched from memory. This has dramatic consequences on the run-time performance, which is now increasingly limited by memory accesses (as opposed to by CPU clock rate). This development has already been predicted in the 90s. However, on modern general-purpose architectures, where programs can process huge and complex data, the processor still can stall a huge fraction of its running time due to the memory gap.

This shows that compiler optimizations for memory accesses are heavily required. However, the imprecise results of alias analyses pose severe restrictions on the optimization. The problem is that the analyses have to consider all possible run-time behavior (whether or not it may actually occur). This directly motivates the use of speculative optimization techniques. The idea is to ignore unlikely dependencies speculatively, which yields more potential for optimization and, eventually, also for performance improvements. Of course, it is important to cope with the case that an ignored dependency *does* occur at run-time. This has to be detected by specific checks, which will trigger in the positive case the execution of so-called *recovery code*, which will re-issue instructions as necessary to ensure program correctness. Since misspeculation poses an overhead, it is important to derive a precise cost model to decide whether or not speculation should be applied at a certain point. This requires precise information about memory dependencies.

To this end, state-of-the-art alias analyses are not sufficient for two reasons: First, as mentioned above, they are too imprecise w.r.t. the false dependencies reported. Second, they only have a binary or at most ternary notion of dependency: absent, maybe present, present. This is too coarse for the optimization. What we need is an analysis that reports the *degree* of a given memory dependency, e.g. as a number in the interval $[0, 1]$. This degree can be thought of as an annotation in the *Data Dependency Graph.* Since we consider speculative optimizations, wrong analysis results are admissible. Hence we can trade off correctness of the analysis for precision.

We propose to use ML techniques to yield the dependency analysis. *Supervised learning* allows us to learn the relationship between *features* or properties of an object and its associated *class.* In our case, the features are the properties of two instructions which may access to the same memory, and the class is the resulting dependency degree (e.g., classes from 0 to 10). In training, a model is built, which represents the encountered data with minimal error. This model can then be used to predict dependencies for new, unseen data. We simply have to collect the training data by profiling, train an ML classifier, and eventually we get an oracle for memory dependencies. This oracle can then be used by our speculative optimization to improve program performance.

We collected the training data for a set of benchmarks and performed pairwise validation, using established ML techniques for classification learning. Preliminary results show that the dependency degree can be predicted with a low error for unseen data. The next step which we are currently working on is to feed this information into the speculative optimization and to measure the improvement gained from the oracle. We expect that this will significantly mitigate the impact of the memory gap, since the latencies of expensive loads will be masked by other instructions.