Abstract

Pointer analysis is one of the key static analyses during compilation. The efficiency of several compiler optimizations and transformations depends directly on the scalability and precision of the underlying pointer analysis. Recent advances still lack an efficient and scalable context-sensitive inclusion-based pointer analysis. In this work, we propose four novel techniques to improve the scalability of context-sensitive points-to analysis for C/C++ programs.

First, we develop an efficient way of storing the approximate points-to information using a multi-dimensional bloom filter (multibloom). By making use of fast hash functions and exploiting the spatial locality of the points-to information, our multibloom-based points-to analysis offers significant savings in both analysis time and memory requirement. Since the representation never resets any bit in the multibloom, no points-to information is ever lost; and the analysis is sound, though approximate. This allows a client to trade off a minimal amount of precision but gain huge savings (two orders less) in memory requirement. By making use of multiple random and independent hash functions, the algorithm also achieves high precision and runs, on an average, $2 \times$ faster than Andersen’s points-to analysis. Using Mod/Ref analysis as a client, we illustrate that the precision is above 98% of that of Andersen’s analysis.

Second, we devise a sound randomized algorithm that processes a group of constraints in a less precise but efficient manner and the remaining constraints in a more precise manner. By randomly choosing different groups of constraints across different runs, the analysis results in different points-to information, each of which is guaranteed to be sound. By joining the results of a few runs, the analysis obtains an approximation that is very close to the one obtained by the more precise analysis and still proves efficient in terms of the analysis time. We instantiate our technique to develop a randomized context-sensitive points-to analysis. By varying the level of randomization, a client of points-to analysis can trade off minimal precision (less than 5%)}
for large gain in efficiency (over 50% reduction in analysis time). We also develop an adaptive version of the randomized algorithm that carefully varies the randomization across different runs to achieve maximum benefit in terms of analysis time and precision without pre-setting the randomization percentage and the number of runs.

Third, we transform the points-to analysis problem into finding a solution to a system of linear equations. By making novel use of prime factorization, we illustrate how to transform complex points-to constraints into a set of linear equations and transform the solution back as a points-to solution. We prove that our algorithm is sound and show that our technique is 1.8× faster than Andersen’s analysis for large benchmarks.

Finally, we observe that the order in which points-to constraints are processed plays a vital role in the algorithm efficiency. We prove that finding an optimal ordering to compute the fixpoint solution is NP-Hard. We then propose a greedy heuristic based on the amount of points-to information computed by a constraint to prioritize the constraints. This results in a dynamic ordering of the constraint evaluation which, in turn, results in skewed evaluation of constraints where each constraint is evaluated repeatedly and different number of times in a single iteration. Our prioritized analysis achieves, on an average, an improvement of 33% over Andersen’s points-to analysis.

We illustrate that our algorithms help in scaling the state-of-the-art pointer analyses. We also believe that the techniques developed would be useful for other program analyses and transformations.