Debugging Optimized Code Via Tailoring

Lori Pollock
University of Delaware
Newark, DE, 19716
pollock@udel.edu

Mary Bivens
Allegheny College
Meadville, PA 16335
mbivens@alleg.edu

Mary Lou Soffa
University of Pittsburgh
Pittsburgh, PA 15260
soffa@pitt.edu

Modern compilers perform ambitious optimization analysis and transformations to produce target code that is time and space efficient. Current techniques to debug optimized programs place restrictions on either the application of optimizations or the types of debugging capabilities provided. Even with the restrictions, the debugging techniques often have to report the existence of noncurrent values. Ideally, we would like to have no restrictions on the debugging capabilities and optimizations, and at the same time, respond to all debugging requests as expected, where expected responses during execution of an optimized program are the same as the responses for an unoptimized version of the same program. However, this is not possible, given the interferences between the optimizations and debugging.

We have developed a scheme to debug optimized programs using analyses developed for incremental optimizations. Optimizations are tailored during debugging to the user's current debugging requests, taking into account both the request and its context within the program. Therefore, there are no a priori restrictions placed on either debugging commands or optimizations. As the user's debugging requests change throughout a debugging session, the optimizations that are allowed in the code also change, eliminating only those that interfere with responses in the expected manner to the user's debugging requests. We handle both machine independent optimizations such as constant propagation, common subexpression elimination, dead code elimination, and loop invariant code motion and target code optimizations including smart register allocation and code selection. In addition to the standard debugging commands such as breakpoint insertion, single stepping, and program state display, we support powerful debugging functions such as changing a variable's value and inserting and deleting code.

In incremental analyses, unnecessary reanalysis and code transformation are eliminated by saving results of prior analyses and using this information to limit the scope of reanalyses. The incremental optimizing techniques include incremental data flow analysis, an incremental intermediate code optimizer, and an incremental target code generator. Both the information saved by these systems and the incremental analyses that they perform have been exploited in our approach to symbolic debugging of optimized code.

In response to debugging requests submitted during execution, the information saved by incremental optimizing analyses is used to determine if the expected response can be ensured, and if not, this information is used to tailor the code to ensure the expected response. In some cases, the code is tailored for the current execution by removing the interfering optimization, servicing the request and continuing with the current execution. When this is not possible, the user is notified of the situation and told that a new execution of the program will produce the expected response to the same debugging request. In this case, the code is tailored by removing the interfering optimization from the next execution. When the programmer knows the debugging requests before execution, the debugger accepts commands prior to an execution and tailors the optimized code using the incremental optimizing techniques to ensure that these planned debugging requests are serviced as expected during execution.

This approach allows the programmer to debug code as close to the final, production code as possible without inhibiting debugging capability. In this system, the user is in control of whether or not to incur the overhead to service a request that is prevented by optimizations. Without the incremental analysis, the tailoring of the optimized code to the current set of debugging commands would be impractical.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association of Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

ISSNA 94 - 8/94 Seattle Washington USA
© 1994 ACM 0-89791-683-2/94/0008..$3.50