Research Interests

Mary Lou Soffa

March, 2004

Optimizing compilers, program analysis, and software tools for debugging and testing programs have been, and continue to be, the focal points of my research interests. The program analysis techniques form the basis for much of compiling techniques and software tools research and thus is the bridge that enables me to span both the programming languages and software engineering communities. I summarize some of my research contributions in this document as well as provide a brief overview of of my current research directions in these areas. The research described was done as joint work with my students and some of colleagues at Pitt and at other universities.

1 Compiler Techniques

My research interests in compiler techniques fall into two categories: code optimizations and instruction level parallelism. Code improving transformations are applied by virtually every compiler today and contribute greatly to the high performance of software. My contributions in compiler optimizations include developing frameworks to explore general properties of optimizations, new approaches for applying optimizations, and improvements in particular optimizations.

Currently, my research in optimizations is taking a novel approach in developing frameworks to identify and explore properties of optimizations. One property of interest is profitability; that is, determining when it is beneficial, or profitable, to apply optimizations. Because of the high cost of applying optimizations and then experimentally evaluating their profitability, we use an analytical approach and develop models for code segments, optimizations, and resources. As part of the framework, we have a profitability engine that uses these models to predict the profitability of applying an optimization at any code point where it is applicable and decides whether or not to apply it accordingly. We have developed models for both loop optimizations and scalar optimizations and resource models for cache, registers and functional units /citezhao:03, /citezhao:04.

I am also continuing the work on path sensitive optimizations. In particular, I am looking at applying optimizations when they can be applied on only certain paths rather than all paths, which is the traditional approach. In one project, we considered the the complete elimination of redundancy, the most effective code transformation applied in compilers and employed both code motion and control flow restructuring to enable path sensitive optimizations as well as speculation. A paper on this result was selected as one of the most influential papers to appear in PLDI /cite Bodik:PRE, Bodik:ICBE, Bodik:LSE.

A seminal piece of work was the development of a technique to specify a class of code improving transformations and the use of this specification to prove properties of optimizations and to automatically implement optimizations [24]. Variations of the specification technique are currently being used in technqiues to prove the correctness of optimizations. With this technique, the enabling and disabling properties of optimizations can be proved and used in developing guidelines for ordering optimizations. A particularly important optimization that we developed and experimentally showed that it was useful was conditional branch elimination that worked both intraprocedurally and interprocedurally [4]. This work also had an application of removing infeasible paths to aid in the precision of testing [5].

In order to further improve on the capacity of code optimizations, we are exploring and developing a framework for both path and resource sensitive optimizations. The machine characteristics, the available resources and the most beneficial paths of a program to optimize, including interprocedural paths, will be factors in the path and resource sensitive optimization algorithms. I plan to also develop techniques that enrich the data flow analysis used by the optimization algorithms with capabilities to exploit machine characteristics and profiling information in a systematic fashion. Another area that I have just started to do research is dynamic optimizations. The question being explored is can the path sensitive optimizations with demand driven analysis be used for dynamic optimizing, and in particular, Java code.

Another type of compiler technique that I have explored in my research is instruction level parallelism. My research in this area has focused on the detection and exploitation of parallelism for architectures such as superscalars and Very Long Instruction Word (VLIW) machines. Region scheduling was one of the first techniques that recognized the importance of integrating control dependence and data dependencies in order to reorder code for scheduling. Importantly, by using region scheduling, the code grow that occurs could be controlled by the user [10]. On of our research projects attacked the phase ordering problem of instruction scheduling and register allocation. The result was a technique called *resource spackling* that more fully integrated the two phases than previous techniques [2, 3].

My work in this area has focused on the prediction of values for use in speculation in scheduling instruction level parallelism. We have shown that using the context of values in predicting can lead to better predictions. That is, we considered various techniques that take into account the path that the program takes [21]. We are also investigating the use of profiling techniques and hardware support for statically scheduled VLIW processors to collect and exploit information for generating instructions schedules that aggressively perform speculation. Currently, we are considering techniques that enable speculation in VLIW machines, which requires compile time compensation code to be produced as well as architectural changes [22].

2 Program Analysis Techniques

Program analysis forms the foundations for the techniques used in many applications today, including code optimizations, scheduling and software engineering applications. My first contribution to program analysis was the development of an interprocedural data flow analysis to compute def-use chairs that could be used during testing. This analysis technique was flow and context sensitive and handled globals, reference parameters and recursive procedures [16].

My current research is exploring techniques to improve on the scalability and precision of data flow analysis. As programs become larger and more complex, the overhead of the analysis techniques increases. An important contribution was the development of demand driven analysis that collects only the data flow information that is needed for a particular application [8]. The demand driven algorithms have proven to be quite useful in reducing the time and space overhead of analysis. Currently, we have as our goal the development of scalable analysis techniques to handle very large applications.

Improving the precision of data flow analysis has been another focal point of research. We developed an analysis technique that detected and eliminated infeasible paths through branch correlation and demonstrated its usefulness in data flow testing [5]. Current interprocedural analyses algorithms typically require the entire program to be in memory, which requires too much memory and time. Thus, another direction of intestest is to develop scalable program analysis for modular or component based software. The idea would be to attach annotations to program modules that describe the results on analysis of a module. These annotations would be combined with other modules when they become part of the program.

3 Testing and Debugging Techniques

Another area of research interest is in the development of software engineer tools, and in particular, testing and debugging. Many of our techniques developed depend heaviliy on program analysis, enabling a cross fertilization between programming languages and software engineering. My work includes techniques for regression testing, generating test cases, slicing, and debugging of optimized code.

In a recent project, we developed a novel way to debug optimizers - by checking values in both the optimized and unoptimized versions of the programs [17] [18]. We also developed a a debugger, FULLDOD, of optimized code that can report more values than any previous techniques /cite JGSa:01,JGS:01,JGS:00. Another area of debugging that we contributed to was that of slicing through the development of a slicing tool that gathers dynamic information as it executes to increase the precision of the information gathered [12]. We are now turning our attention to debugging of dynamically applied optimizations /cite SKVC:03.

I have contributed to a number of different facets of software testing. Our current work involves testing the interfaces between applications and the databases it uses. Previously, testing has been done separately on the application and the databases but the interface was not tested. In this work, we are developing a framework for testing of interfaces. To date, we have developed adequacy metrics for testing these interfaces; that is, measures that determine the completeness of the testing. A paper on this recently was awarded a best paper at the 2003 Foundations of Software Engineering cite /Kapf:03.

Recently, we developed the first testing framework for Graphical User Interfaces (GUI) which included novel adequacy measures, test case generation strategies, oracles and an executor that automatically tests the GUI. A paper on this work received a best paper award at Foundations of Software Engineering /cite MPS:00, Mem:03, MSP:01.

Regression testing involves determining what parts of a program to retest or what test cases must to rerun to retest a program after modifications are made to the program. Most of my work has been based on data flow testing techniques. We developed incremental techniques that identified effected data-flow associations and updated the data flow changes as well as determining the tests that were effected [15]. One of the techniques was demand driven which was more efficient both in terms of time and memory [9] [13].

Test case generation is another area that I have made a contribution. Some of my research has had as its goal the reduction of test case suites [14] [11]. A current project has as its goals the development of automatic techniques for testing Graphical User Interfaces (GUI). GUIs are becoming an important and accepted way to interact with today's software and currently we have few systematic ways to test them. We are approaching this problem in a novel way by using AI planning techniques. So far, we have been able to demonstrate the value of AI planning test case generation. Our current work includes the development of testing oracles, coverage measures and regression techniques for GUIs [19] [20].

References

[1] Automated test oracles for guis. In *Eighth International Symposium on the Foundations of Software Engineering*, 2000.

[2] D. Berson, R. Gupta, and M.L. Soffa. Resource spackling: A framework for integrating register allocation in local and global schedulers. In *IFIP Proceedings of International Conference on Parallel Architectures and Compilation Techniques*, pages 135–146, Montreal, Canada, 1994.

[3] D. Berson, R. Gupta, and M.L. Soffa. Integrated instruction scheduling and register allocation techniques. In *Proceedings of International Workshop on Languages and Compilers for Parallel Computing*, Springer Verlag, Chapel Hill, North Carolina, 1998.

[4] Rastislav Bodik, Rajiv Gupta, and Mary Lou Soffa. Interprocedural conditional branch elimination. In *Proceedings of Programming Languages Design and Implementation (PLDI)*, pages 146–158, June 1997.

[5] Rastislav Bodik, Rajiv Gupta, and Mary Lou Soffa. Refining data flow information using infeasible paths. In *ACM SIGSOFT Symposium on Foundations of Software Engineering*. LNCS Nr. 1301, Springer–Verlag, September 1997.

[6] Rastislav Bodik, Rajiv Gupta, and Mary Lou Soffa. Complete removal of redundant expressions. In *Proceedings of Programming Languages Design and Implementation (PLDI)*, pages 1–14, June 1998.

[7] Rastislav Bodik, Rajiv Gupta, and Mary Lou Soffa. Load-reuse analysis: Design and evaluation. In *Proceedings of Programming Languages Design and Implementation (PLDI)*, May 1998.

[8] Evelyn Duesterwald, Rajiv Gupta, and Mary Lou Soffa. A practical framework for demand-driven interprocedural data flow analysis. *ACM Transactions on Programming Languages and Systems*, to appear.

[9] R. Gupta, M.J. Harrold, and M.L. Soffa. Program slicing-based regression testing techniques. *Journal of Software Testing, Verification and Reliability*, June 1996.

[10] R. Gupta and M.L. Soffa. Region scheduling: An approach for detecting and redistributing parallelism. *IEEE Transactions on Software Engineering*, pages 421–431, April 1990.

[11] R. Gupta, M.L. Soffa, and J. Howard. Employing static information in the generation of test cases. 1993.

[12] R. Gupta, M.L. Soffa, and J. Howard. Hybrid slicing: Integrating dynamic information with static analysis. *ACM Transactions of Software Engineering Methodology*, pages 370–397, 1997.

[13] Rajiv Gupta, M.J. Harrold, and M.L. Soffa. An approach to regression tesing using slicing. In *Proceedings of IEEE/ACM Conference on Software Maintenance*, pages 299–308, 1992.

[14] M. J. Harrold, R. Gupta, and M.L. Soffa. Methodology for controlling the size of a test suite. *ACM Transactions on Softwrae Engineering and Methodology*, July 1993.

[15] M.J. Harrold and M.L. Soffa. An incremental approach to unit testing during maintenance. In *Proceedings of IEEE/ACM Confernce on Software Maintenance*, pages 362–367, 1988.

[16] M.J. Harrold and M.L. Soffa. Efficient computation of interprocedural definition-use chains. *ACM Transactions of Programming Languages and Systems*, pages 175–204, March 1994.

[17] C. Jaramillo, R. Gupta, and M.L. Soffa. Capturing the effects of code improving transformations. In *Proceedings of International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 118–123, 1998.

[18] C. Jaramillo, R. Gupta, and M.L. Soffa. Comparison checking: An approach to avoid debugging of optimized code. In *Proceedings of the Joint 7th European Software Engineenering Conference and the ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, pages 268–284, Toulouse, France, 1999.

[19] A. Memon, M. Pollack, and M.L. Soffa. Using a goal-driven approach to generate test cases for guis. In *Proceedings of International Conference on Software Engineering*, pages 257–266, Los Angeles, 1999.

[20] A. Memon, M. Pollack, and M.L. Soffa. Plan generation for gui testing. In *The Fifth International Conference on Artificial Intelligence Planning and Scheduling*, Brackenridge, Co, 2000.

[21] T. Nakra, R. Gupta, and M.L. Soffa. Global context-based value predition. In *Proceedings of the Fifth International Symposium on High Performance Computer Architecture*, pages 4–12, Orlando, Florida, January 1999.

[22] T. Nakra, R. Gupta, and M.L. Soffa. Value prediction in vliw machines. In *ACM/IEEE 26th International Symposium on Computer Architectures (ISCA)*, pages 258–269, Atlanta, Georgia, 1999.

[23] K. Scott, N. Kumar, S. Verlusamy, B. Childers, J. Davidson, and M.L. Soffa. Retargetable and reconfigurable software dynamic translation. In *Conference on Code Generation and Optimization*, 2003.

[24] D.L. Whitfield and Mary Lou Soffa. An approach for exploring code-improving transformations. *ACM Transactions on Programming Languages and Systems*, November 1997.

[25] Min Zhao, Bruce Childers, and Mary Lou Soffa. Profit driven optimizations. In *submitted*.

[26] Min Zhao, Bruce Childers, and Mary Lou Soffa. Predicting the impact of optimizations for embedded systems. In *ACM SIGPLAN Conference on Languages, Compilers, and Tools for Embedded Systems*, 2003.