A GSoC 2013 Proposal

Enhancing Giri: Dynamic Slicing in LLVM

Mingliang Liu, *Tsinghua University*

Dynamic program slicing has been used in many applications. Giri was a research project from UIUC, which implemented the dynamic backward slicing in LLVM. I think it's a good idea to extend this project in several ways: *1*) Update the code to LLVM mainline and make it robust, *2*) Improve the performance of giri run-time, *3*) Reduce the trace size, etc.

Background

Program slice contains all statements in a program that directly or indirectly act the value of a variable occurrence [14], the criteria of which is a pair of statement and variables. We can further narrow the notion of *slice*, which contains statements that influence the value of a variable occurrence for special program inputs. This is referred as dynamic program slicing [1]. It works on a single execution and outputs the executed statements (traces) relevant to the slicing criterion.

There are many applications that use (or could benefit from) dynamic slicing, both by research and industry organizations (e.g. Microsoft, IBM). For example, it's long been used in software debugging model [2, 5] and testing [3]. Sahoo et. al. from UIUC use dynamic program slicing to generate likely invariants for automated software fault localization [10]. Differential slicing [6] was a joint work by UC Berkeley, IMDEA Software Institute and CMU. It uses dynamic slicing to establish the sequence of value differences that affect the target, which can help them identifying causal execution differences for security applications. Gupta et. al. from the University of Arizona employed dynamic program slicing to narrow down the search for faulty code [4]. Our group from Tsinghua University would benefit from dynamic slicing in automatically finding manual configuration errors [15] in software deployment (see below). Recently, researchers from National University of Singapore and Microsoft used dynamic slicing to debug evolving programs [8]. Dennis Jeffrey from Google built a system for debugging via online tracing and dynamic slicing with other guys from university [7]. Researchers from IBM Research[9] who work on fault localization for data-centric programs, split the trace into multiple slices by applying dynamic slicing.

Motivation

There are two projects public available which implement the program slicing in LLVM. LLVMSlicer [11] implementation is a static backwards slicer from Masaryk University. It works on the well defined data and control flow equations in a white paper by F. Tip [13]. Giri was a research project from UIUC, which implemented the static and dynamic backward slicing. It also maps LLVM IR statements to source-level statements for its output using the debug metadata. The developers of Giri are active in LLVM community and willing to release their code. I think improving the Giri dynamic slicing code would be a good idea under their kind direction.

Our goal is to release the Giri code as a sub-project of LLVM. As far as I know, there is no public available dynamic slicing tool in GCC or Open64. We choose LLVM because its scalar variables are kept in well-defined static single assignment (SSA) form, making definition-use chains explicit. As to the tool chain of Giri code, they currently link the Giri passes into libLTO and run on the whole program bitcode before libLTO generates native code. This can also ensure that each instrumented instruction gets its own unique ID.

Plan

There are several things I can do for Giri in this summer of code.

1. Updating the code to LLVM mainline and putting it into the giri SVN repository
2. Reducing the trace size. Giri currently records the execution of each basic block, load and store, call, and return instruction. There are things I can do to make the trace smaller (e.g., creating one trace record for control-equivalent basic blocks).
3. Making the giri run-time library thread safe and the slicing thread/process aware. Right now, I think the events of all processes and threads get thrown together in one trace file. Each should have a separate trace file, or trace records should indicate which thread is performing a particular operation.
4. Improving support to correctly handle asynchronous events (e.g., signal handlers).
5. Improving the giri run-time performance. The current run-time library `mmaps` a portion of the trace file into memory, writes trace records into it, and then synchronously `munmaps` it and then maps in the next portion of the trace file. This design ensures that we don't swamp memory (the application can produce trace records faster than the OS can write them to disk), but it doesn't overlap computation with I/O very well. The run-time also has a static size for how many trace records to hold in memory before flushing to disk; this value should be computed dynamically.
6. Handling external library calls which is not complete for some calls.

Table 1: *Plan of the project*

Work	Weeks
Make Giri up to date	1
Profile the code	1
Make run-time library thread safe	2
Handle asynchronous events	2
Reduce trace size	2
Improve the giri rum-time performance	3
Improve source code generating	1
Scrub code, write tests	1

About Me

I'd like to introduce myself briefly. I'm a three-years PhD candidate student from Tsinghua University, China. My research area covers performance analysis, compiler techniques for high performance computing, and parallel computing (MPI/OpenMP).

One of my on-going work is to generate an I/O benchmark from the original application. The base observation is that the computation and communication statements can be deleted if they're irrelevant to the I/O pattern, e.g. computing the buffer content to be written into a file. We take use of the program slicing technique to find relevant/irrelevant statements. The static slicer was borrowed from LLVMSlicer and I wrote code to make it work for our application. We generated the line number of sliced code. There is a very simple source code generation script using ugly and tricky regular expressions to delete original source code according to the sliced line number. We're going to submit the first version of the paper recently.

I also took part in one project in Open64 compiler several year ago, the purpose of which is to fast collect the communication trace. We used program slicing to delete the computation statements and kept the communication related statements. We generated executable binaries instead of source code from IR.

Now we plan to do a project that can automatically find manual configuration errors [15] in software deployment. The dynamic slicing can help a lot since the input is the key factor to locate errors. We have not a concrete plan for this project, but the dynamic slicing is heavily needed. Our long-term plan is to add more features, e.g. Objective-C/C++ support, thing slicing [12].

My email is liuml07@gmail.com. My homepage is at http://pacman.cs.tsinghua.edu.cn/~liuml07

References

[1] H AGRAWAL. Dynamic Program Slicing. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 1990.

[2] Hiralal Agrawal, Richard A Demillo, and Eugene H Spafford. Debugging with dynamic slicing and backtracking. *Software: Practice and Experience*, 23(6):589–616, 1993.

[3] Hiralal Agrawal, Joseph R Horgan, Edward W Krauser, and Saul A London. Incremental regression testing. In *Software Maintenance, 1993. CSM-93, Proceedings., Conference on*, pages 348–357. IEEE, 1993.

[4] Neelam Gupta, Haifeng He, Xiangyu Zhang, and Rajiv Gupta. Locating faulty code using failure-inducing chops. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, ASE '05, pages 263–272, New York, NY, USA, 2005. ACM.

[5] Tibor Gyimóthy, Árpád Beszédes, and István Forgács. An efficient relevant slicing method for debugging. In *Software EngineeringESEC/FSE99*, pages 303–321. Springer, 1999.

[6] Noah M Johnson, Juan Caballero, Kevin Zhijie Chen, Stephen McCamant, Pongsin Poosankam, Daniel Reynaud, and Dawn Song. Differential slicing: Identifying causal execution differences for security applications. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 347–362. IEEE, 2011.

[7] Vijay Nagarajan, Dennis Jeffrey, Rajiv Gupta, and Neelam Gupta. A system for debugging via online tracing and dynamic slicing. *Software: Practice and Experience*, 42(8):995–1014, 2012.

[8] Dawei Qi, Abhik Roychoudhury, Zhenkai Liang, and Kapil Vaswani. Darwin: An approach to debugging evolving programs. *ACM Trans. Softw. Eng. Methodol.*, 21(3):19:1–19:29, July 2012.

[9] Diptikalyan Saha, Mangala Gowri Nanda, Pankaj Dhoolia, V. Krishna Nandivada, Vibha Sinha, and Satish Chandra. Fault localization for data-centric programs. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, ESEC/FSE '11, pages 157–167, New York, NY, USA, 2011. ACM.

[10] Swarup Kumar Sahoo, John Criswell, Chase Geigle, and Vikram Adve. Using likely invariants for automated software fault localization. In *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems*, ASPLOS '13, pages 139–152, New York, NY, USA, 2013. ACM.

[11] Jiri Slaby. Static Slicer for LLVM. `https://github.com/jirislaby/LLVMSlicer`, 2013.

[12] M. Sridharan, S.J. Fink, and R. Bodik. Thin slicing. In *Proceedings of the ACM SIGPLAN conference on Programming Language Design and Implementation*, volume 10 of *PLDI'07*, pages 112–122, 2007.

[13] F. Tip. A survey of program slicing techniques. *Journal of Programming Languages*, 3(3):121–189, 1995.

[14] M. Weiser. Program slicing. In *Proceedings of the 5th International Conference on Software Engineering*, ICSE, pages 439–449. IEEE, 1981.

[15] Zuoning Yin, Xiao Ma, Jing Zheng, Yuanyuan Zhou, Lakshmi N Bairavasundaram, and Shankar Pasupathy. An Empirical Study on Configuration Errors in Commercial and Open Source Systems. In *SOSP*. ACM, 2011.