A GSoC 2013 Final Report

Enhancing Giri: Dynamic Slicing in LLVM

Mingliang Liu, Swarup Kumar Sahoo
http://github.com/liuml07/giri

Dynamic program slicing has been used in many applications. Giri was a research project from UIUC, which implemented the dynamic backward slicing in LLVM. It was selected as the Google Summer of Code 2013. We achieved several improvements to Giri during GSoC 2013: *1*) Update the code to LLVM mainline and make it robust, *2*) Reduce the trace size, *3*) Make the Giri thread-aware (pthread only), *4*) Improve the performance of run-time.

1 Introduction

Dynamic program slice contains statements that influence the value of a variable/instruction occurrence for given program inputs [1]. The traditional program slicing was called static program slicing, which was firstly proposed by Weiser [7]. 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 [2, 4] and testing [3]. However, as far as we're concerned, there is no publicly available dynamic slicing tool in either GCC or Open64.

Sahoo et. al. from UIUC use dynamic program slicing to generate likely invariants for automated software fault localization [5]. They implemented the dynamic program slicing code, called Giri, in LLVM compiler infrastructure (version 2.6) for research purpose. It traces the user program execution and reports the dynamic slices in the end. It also maps LLVM IR statements to source code for its output using the debug metadata. To make the Giri up to date and robust for general usage, we enhanced its code in several factors.

The report is organized as following. Section 2.1 shows the internal architecture of Giri, including how it works and how to use it. Section 3 lists the progress we made during the GSoC 2013. Section 4 is the TODO list of Giri as future work. Section 5 concludes our work, and attaches our project URL.

2 Overview

2.1 Dynamic Program Slicing

The following is an example program from the paper of dynamic program slicing [1].

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int x, y, z;

    x = atoi(argv[1]);

    if (x < 0)
    {
        y = sqrt(x);
        z = pow(2, x);
```

```
15.     } else {
16.         if (x == 0)
17.         {
18.             y = sqrt(x * 2);
19.             z = pow(3, x);
20.         } else {
21.             y = sqrt(x * 3);
22.             z = pow(4, x);
23.         }
24.     }
25.
26.     printf("%d\n", y);
27.     printf("%d\n", z);
28.
29.     return z;
30.}
```

Given the input `x = 10`, the source line numbers affect the return value are: *<9, 22, 29>*.

The default slicing criterion is the `ret` instruction at the `main` function. Note that there is only one `return` when the main executes. Giri also supports two more ways to specify the slicing criterion, which are source code line number and LLVM instruction number respectively. See the `test/UnitTests/test4` for more information.

2.2 The Design of Giri

Giri handles both data-flow and control-flow dependences when computing the dynamic backwards slice. It has two phases, in the first which (*tracing* pass), it instruments the code to record LLVM IR in a trace file at run-time for all threads, and in the other of which (*slicing* pass), it uses the execution trace to create a program dependence graph to find dynamic slice.

The tracing pass instruments code to record three different pieces of information: (a) basic block exits; (b) memory accesses and their addresses; and (c) function calls and returns. When the slicing pass first adds instructions from a dynamic execution of a basic block to the backwards slice, it uses static control-dependence analysis to determine which value forced the execution of that basic block. Giri will then find the most recent execution of the instruction generating that value and add it to the backwards slice. The slicing pass ends when all dynamic values are processed and reports the dynamic slice of the specific program execution.

To make the slicing pass thread aware, the runtime library traces the thread id (`thread_t`) to each record indicating the very thread doing an operation. The slicing pass checks the thread id when scanning the trace file for a given dynamic value or basic block.

3 Progress

Our goal of GSoC was to make the Giri code update to the latest LLVM version, improve its runtime performance, and/or reduce the tracing cost. There are several things we did in this summer:

1. **Update the code to LLVM mainline.**
   - Release a v3.1 version which works with LLVM 3.1
   - Make the code update to the latest LLVM 3.4 (`r191529`)
2. **Make the Giri run-time library thread safe and the slicing pass thread aware.** The operation traces of all threads are written to a single file. Trace records include thread id (`pthread_t`), which indicates the thread performing a particular operation. We use locks when write traces to file in order to avoid the race condition.
3. **Improve the Giri run-time performance.** We fixed the bug of `mmap` function call at the end of the tracing. Giri now dynamically computes the cache size of trace records to hold in memory before flushing to disk.
4. **Write dozens of unit tests and try several real programs.** There is also a simple test framework which runs all the unit tests at the top level of `giri/test/` and report the result. In the future development, every patch should be checked before committing it to the git repository.
5. **Reduce the trace size.** We truncate the file at the end of tracer according to its real size.
6. **Write documents for the project.** There are several sample pages:
   - The Wiki Page Home.
   - How to Compile Giri.
   - Hello World!
   - Example Usage With An Example.

4 TODO List

The following are the TODO list we're going to address in the next few months. An update one is maintained online at the wiki page of Giri at github.com.

1. Improve the performance of locking mechanism at the runtime
2. Write more unit tests with complicated direct/indirect recursive function calls
3. Double check the `TraceFile.cpp` to make sure the thread id be checked correctly
4. Add `tool/Tracer.cpp` to the make list. It was removed when upgrading to LLVM 3.4
5. Pass more real world programs and add them to the `test/` directory
6. Try large test programs, e.g. *nginx*, *squid*, etc
7. Make Giri code runnable at other platforms besides Linux, e.g. Mac OS X, Cygwin, and FreeBSD.
8. Consider more special function calls in `TracingNoGiri::visitSpecialCall()` function of the tracing pass
9. Parallelize the code writing the entry cache to trace file and adding entry to the cache.

5 Conclusion

As the first publicly available dynamic slicing tool, Giri was made better during the GSoC. We updated the code to LLVM mainline, reduced the trace size, made it thread-aware, and improve the performance of its run-time. We publish our code at https://github.com/liuml07/giri [6].

The code is still under active development. Dr. Swarup will direct Mingliang LIU in the future to make the Giri code better. There are other guys from Tsinghua University, Xi'an Jiaotong University and University of Miami, who wrote to us in email hoping to use and contribute to Giri project. For more information, please visit the homepage above.

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

[5] 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.

[6] Swarup Kumar Sahoo, Criswell John, and Mingliang LIU. Giri: Dynamic Program Slicing in LLVM. `https://github.com/liuml07/giri`, 2013.

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