Building and Executing Traces on LLVM

Daniel Nicacio
IC-UNICAMP, Brazil

April 5, 2011

1 Objective

The objective of this project is to augment LLVM with dynamic profiling capabilities. Interpreted code will be instrumented so that LLVM will be able to dynamically find, build and execute traces of more frequently executed basic blocks. This way, future projects can focus on implementing dynamic optimizations on those traces.

2 Motivation

Just In Time (JIT) compilers and Dynamic Binary Translators (DBT) can provide information collected at runtime. The static compiler does not have this kind of information, so using it, new optimization opportunities comes to surface. In order to have a efficient optimization system, it is essential to know where to apply optimizations, that is, know how to obtain critic regions of code which really sets out gain possibilities. A sequence of instructions, including branches but not including loops, that is executed for some input data is called a trace [5]. Dynamic systems can select traces which are heavily executed as good spots to make optimizations. Optimizing traces is one promising way to overcome the overhead of translating instructions to the new ISA, permitting the new code to run faster than the original one.

In the last years, dynamic optimizations has become an interesting subject due to its many advantages, being the research topic of some works [1, 3, 4, 8]. All these works must rely on good trace detection techniques.

At the present moment LLVM does not build and execute traces using runtime information. We believe that a light-weight dynamic profiling will be a valuable tool to improve LLVM performance.

3 Methodology

In this project we will implement the Most Recent Executed Tail (MRET) [2] technique to find and build traces. On the first stage we will find basic blocks

that are targets of back edges (potential loop headers) and then instrument those basic blocks to count how many times it is executed. Therefore, if it reaches a threshold a function call (previously added to the basic block) is executed to start recording the final trace. Finally the recording process ends when it goes back to the beginning of the trace or reaches another trace.

Every time program execution reaches an address that corresponds to the beggining of a trace it jumps to the trace instead of the original basic block. Traces are stored together in memory; this fact alone already brings benefits to the system, because the program execution will likely remains inside this region of memory, taking advantage of the locality principle when fetching instructions.

4 Timeline

This project is expected to last three months, divided as follows:

- **first month** - instrument basic blocks to count how many times it is executed and add a function call to trace recording.
- **second month** - Record traces and change original code so it can jump to traces instead of the original basic blocks.
- **third month** - Measure performance of llvm when executing traces and compare it to the original implementation. Make some tuning if necessary.

5 Biography

I am a PhD student at the University of Campinas (UNICAMP), Brazil. My advisor is Guido Araujo, who has major experience in compilers and computer architecture. During my research I worked with dynamic binary translators, developing trace detection techniques and dynamic optimizations to be applied on those traces. In this project I found out that building traces does not add significant overhead and that code reallocation brings performance gains to the system.

I also worked with a dynamic binary translator that used a direct mapping technique to run PowerPC code on X86 architectures, this project also granted me experience on dynamic environments and code instrumentation.

More recently, I have been working with Software Transactional Memory (STM), building a light-weight user-level transaction scheduler. We achieved great results in this project, pushing the boundaries of STM performance. Basicaly, we implemented a heuristic to predict transaction conflicts, if we find that a transaction is likely going to abort, we switch contexts at user-level to start executing another transaction. One major advantage of our system is that we have complete control of which transactions are currently running and which transaction is going to replace a doomed one.

My research has yielded two papers already published [6, 7] and I also have two papers under review at the present moment.

I hope this project gets accepted because I believe that it will be useful for the LLVM community. It would be a wonderful experience to work alongside LLVM developers and to learn some practical aspects of coding for a robust compiler such as LLVM.

References

[1] V. Bala, E. Duesterwald, and S. Banerjia. Dynamo: A transparent Dynamic Optimization System. *SIGPLAN PLDI*, pages 1–12, June 2000.

[2] Vasanth Bala, Evelyn Duesterwald, and Sanjeev Banerjia. Dynamo: a transparent dynamic optimization system. *SIGPLAN Not.*, 35(5):1–12, 2000.

[3] Kemal Ebcioğlu and Erik R. Altman. Daisy: dynamic compilation for 100 In *ISCA '97: Proceedings of the 24th annual international symposium on Computer architecture*, pages 26–37, New York, NY, USA, 1997. ACM.

[4] J. Lu, H. Chen, P.-C. Yew, , and W.-C. Hsu. Design and implementation of a lightweight dynamic optimization system. *The Journal of Instruction-Level Parallelism*, 6, 2004.

[5] Steven S. Muchnick. *Advanced Compiler Design and Implementation*. Morgan Kaufmann Publishers, 340 Pine Street , Sixth Floor, San Francisco , CA 94104-3205 , USA, 1997.

[6] Daniel Niccio and Guido Arajo. Reducing false aborts in stm systems. In Ching-Hsien Hsu, Laurence Yang, Jong Park, and Sang-Soo Yeo, editors, *Algorithms and Architectures for Parallel Processing*, volume 6081 of *Lecture Notes in Computer Science*, pages 499–510. Springer Berlin / Heidelberg, 2010.

[7] Maxwell Souza, Daniel Nicácio, and Guido Araújo. ISAMAP: Instruction Mapping Driven by Dynamic Binary Translation. In Mauricio Breternitz, Robert Cohn, Erik Altman, and Youfeng Wu, editors, *AMAS-BT - 3rd Workshop on Architectural and Microarchitectural Support for Binary Translation*, Saint Malo France, 2010.

[8] D. Ung and C. Cifuentes. Optimising hot paths in a dynamic binary translator. In Workshop on Binary Translation, October 2000.