LLVM: Improve Debugging of Optimized Code
GSoC 2018 Application

Grammenos Anastasis
e-mail: anastasis.gramm2@gmail.com
irc: gramanas

About me

My name is Anastasis Grammenos. I am a undergraduate student at CSd AUTh, in Thessaloniki, Greece. Last year I successfully completed GSoC with Mixxx, you can check out my project here. I have experience with C++ and git and I like using/configuring Linux. I also know some bash, a bit of python and latex, and the very basics of elisp, Octave and gnuplot.

I've been a "developer" for the last 3 years mainly by writing/maintaining scripts and simple programs in python and batch/bash, in order to automate simple tasks in a photography shop I used to work. In the spirit of photography I am currently developing a simple utility application, rrwc, in C++ and Qt. Since my introduction to the open source developers community with last year's GSoC, I have really enjoyed my stay. I like contributing and using software made by us, for us.

I use Emacs for all my editing and I can also work remotely via ssh.

About LLVM

Compilers, to me, are one of the most interesting pieces of software. After spending the last weeks getting familiar with the project's structure, I can safely say that I like LLVM even more. The modularity of the codebase and the whole dev process are highly intriguing as well.

I have taken a "Theory of Computation" and "Intro to Language Design" courses in the university and although the slides for the last one were so old that they claimed that C++ doesn't support threads, I really enjoyed those lessons and if I ever go for a Master's degree it will probably be in this field or related.

Current involvement with llvm

I have a working LLVM development environment set up through emacs.

After I went through the kaleidoscope tutorial, I started looking at the code under AsmParser and DebugInfo while investigating this bug (34562) that was suggested to me at the mailing list. Since the project I would like to work on is about DebugInfo I believe it's a good place to start.

This bug turned out to be a dead end for now, but it helped me a lot discovering the structure of the opt tool and the IR parser.

Also I committed a minor patch related with DebugInfo.

Improve Debugging of Optimized Code

After looking through the suggested projects, this one piqued my interest. Debugging an application is something we do quite a lot, and optimizations shouldn't make this harder. Improving optimized debugging, results in more precise stack traces and makes the `frame variable` operation work better.

General Notes

As the suggestion mentions in the `LLVM` projects page, this project has two goals. At first I'll have to gather some metrics about the `DebugInfo` lost from various optimization passes. Then I'll start fixing various `DI` loss occurrences. The areas of the compiler that are used the most will be prioritized.

The `debugify` pass

The `debugify` pass attaches consecutive debug locations and values to all the instructions in a `llvm:Module`. The generated `DI` is synthetic, meaning it doesn't describe any real correspondence between instructions and a source file. It has some useful properties like a unique line location for each instruction and a unique variable for each instruction with a value.

Collecting statistics about `DebugInfo` loss

The first goal requires a creation of a new mode: `debugify-each`. It will allow the opt tool to gather metrics about `DI` lost. We'd run `debugify`, then a pass, then `check-debugify`, store the `DI` loss statistics and repeat for each pass we want to check.

Running this with `opt -O{1,2,3}` in samples of `bitcode` will allow us to pinpoint where the `DI` loss is occurring.

Addressing `DI` loss bugs

After that, I'll be incrementally fixing the `DI` loss problems, starting from the most used ones, while checking with the `debugify-each` mode that the results are getting better.

Just like in this patch, a transformation could be killing an instruction and with it, valuable debug information. A `salvageDebugInfo()` call before the deletion should take care of preserving said debug info.

Resources

All these tests will require a handful of source code files to check were `DI` loss is occurring. I plan to use `Mixxx` source code, since I already build it using `llvm`, as well as some big source files from `llvm` like `X86ISelLowering.cpp`. I could also use amalgamated SQLite sources.

As for the visualization of the data, I will probably use `GnuPlot`.

Timeline

Coding officially begins in May 14.

Week 1:	May	Introduce a dummy `debugify-each` mode to opt, just like `verify-each`. Add tests to `debugify-each` option.
Week 2:		Teach `debugify-each` mode to apply and validate debugify metadata. Update the tests. Implement a way for `debugify-each` mode to export the `DI` loss results.
Week 3:		Report `DI` loss statistics for each pass in the `-O1`, `-O2` and `-O3` pipelines. Chart the data to provide visual feedback. Attempt to make inferences about differences between `C/C++` sources, or about where the most critical `DI` loss bugs lie.
Week 4:	June	Pick a pass with a high rate of DI loss and narrow down a single, culprit transform. Devise a patch to preserve debug info in this transform.
Week 5-8:		Repeat the previous process for other passes. Monitor the `DI` loss statistics collected by debugify-each and at the end of each week inform the community of the results.
Week 9:	July	Since in June I'll have finals in the university I'm gonna need a week to catch up with the schedule.
Week 10-12:		Test end-to-end cases. Compile real/specially crafted programs and debug them. When values go missing or are optimized out, identify the pass responsible and try to fix it.
Final week:	August	Finish.

Extra

I would like to add documentation in the `Source Level Debugging` page about the `debugify` pass and the `debugify-each` mode I will create. This can be done in conjunction with the rest of the work.

Another thing worth pointing out, is that I will be keeping a weekly devlog `here`. At the end of each week I'll update it with the relevant information about my project's progress.