Critique on ”LLVM: A compilation framework for lifelong program analysis & transformation” [1]

Junghoon Seo
GIST

April 9, 2017

1 Summary

This paper introduces the *LLVM* compiler frame work for transparent, life-long program analysis and transformation. The two elements of LLVM, program representation and compiler architecture, are described in detail as follows. The LLVM code representation has the following key features. (A) A low-level, language-independent type system, (b) Type conversion instruction preserving type information (c) Two low-level exception-handling instructions. The LLVM compiler has a wide variety of components such as external front-end & static optimizers, compile- and link-time optimizers, and offline or JIT native code generators to achieve the stated goals. Both experimentally and conceptually, LLVM provides a low-level but rich, compat, fast, retargetable, and retrospective representation and compile system.

2 Strength

LLVM is considered to have succeeded in binding the compiler environment, which was divided into a static compiler and a runtime / JIT compiler. Apart from its excellence, many sub-projects of LLVM, such as *clang*, show a great deal of contribution from this paper.

3 Weakness

There is a somewhat lack of examples of code representation and type conversion in the paper. However, it is a fairly understandable part of the feature of the subject that the paper describes, which could be tricky like the API docs if authors start to write in detail.

4 Conclusion

LLVM is designed for compile-time, run-time, and static optimization for any language. LLVM provides a complete middle compiler layer using IR generated by the compiler and optimized IR. LLVM also generates relocatable naive code at run-time or at compile/link-time.

References

[1] Chris Lattner and Vikram Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization*, page 75. IEEE Computer Society, 2004.