A Llvm Backend for a Just-In-Time (Jit) Compilation Engine of a state-of-the-art Instruction Set Simulator (Iss)

University of Edinburgh, School of Informatics

Abstract

Instruction Set Simulators as well as Virtual Machines implement JIT compilation techniques for improving the runtime performance of computer programs. The idea is to convert code compiled for some different architecture into native machine code at runtime to increase simulation speed. As part of the PASTA project a state-of-the-art ISS for the ARC© instruction set architecture implementing JIT compilation techniques has already been developed. The aim of this project is to extend the current JIT compilation engine with a LLVM (Low Level Virtual Machine) backend to decrease JIT compilation time whilst generating fast code.

ArcSim is a state-of-art high-speed Iss that supports Jit compilation techniques and has been developed within the Pasta project. It translates binary code compiled for the Arc© instruction set architecture into native code at runtime. First the simulator starts to execute a binary in interpretive mode and collects statistics about basic block execution frequencies in order to determine when to translate a basic block into native code. When a basic block becomes sufficiently "hot" (i.e. it has been executed frequently) the Jit compiler generates native code for it. At the moment this works by generating highly efficient C code that is then compiled to the native system instruction set architecture (i.e. `x86`). Essentially this means that C is our intermediate representation (Ir) for Jit compilation. While this approach is highly portable, common practice (e.g. the functional language Haskell used and still supports a similar approach for code generation), and generates highly efficient code, Jit compilation times are higher because of the overheads introduced by using C as an Ir.

By using C as an Ir and Gcc as a Jit compiler, compilation times suffer from the following problems:

- Lexical analysis, parsing, and context sensitive analysis (i.e. type

checking) usually represent the largest fraction of compilation time.

- Register allocation consumes a significant fraction of compilation time, especially for large "hot" blocks where register pressure is high.
- Depending on the JIT compilers optimisation options other optimisation passes such as common subexpression elimination (CSE), alias analysis, construction of Static-Single-Assignment (SSA) form and SSA optimisations etc. contribute towards increased compilation times.

Instead of using GCC as a JIT compiler we could use TINYCC. TINYCC is faster than GCC when compiling C code without optimisations turned on, but does not implement optimisation options that are necessary to enable fast simulation speeds.

The main aim of this project is to implement a second JIT engine backend that is capable of directly emitting LLVM bitcode that can in turn be compiled to native code using the LLVM compiler. The benefits of this approach are as follows:

- By generating LLVM bitcode directly instead of C code, the time consumed by lexical analysis and parsing is drastically reduced.
- The LLVM IR resembles a RISC like instruction set. Thus the mapping of ARC© RISC instructions onto LLVM RISC instructions should be straight forward.
- The LLVM compiler implements an efficient linear scan register allocation algorithm that is faster than traditional graph colouring based algorithms and still produces good code (e.g. the Java HotSpot JIT compiler also uses a linear scan register allocation algorithm [3]). This should further decrease compilation times for large JIT compiled basic blocks.
- The LLVM compiler provides a wealth of optimisation passes capable of generating very fast code.

Finally a comparison of achieved compilation speeds against the baseline C JIT backend using standard benchmarks and custom compute and data intensive applications (e.g. AAC decoding, fractal computations) should be performed. A comparison with another state-of-the-art instruction set simulator with respect to JIT compilation speed would be desirable but is not strictly necessary for the purpose of this project.

Try to design your solution to the proposed problems with simplicity and easy readability in mind. It is essential to pay attention to good programming style as well as good documentation. After all it should be easy for other persons to maintain and improve your code later on. The resulting code will carry a simple and permissive open source license (e.g. BSD license).

Supervisors: Björn Franke (bfranke@inf.ed.ac.uk), Nigel Topham (npt@inf.ed.ac.uk)

References

[1] Chris Lattner. LLVM: An Infrastructure for Multi-Stage Optimization. Computer Science Dept., University of Illinois at Urbana-Champaign, 2002.

[2] LLVM Website: http://llvm.org

[3] Christian Wimmer. Linear Scan Register Allocation for the Java HotSpot Client Compiler. Master's thesis, Johannes Kepler University Linz, August 2004.