Concurrent, Adaptive, Just-In-Time (JIT) and Ahead-Of-Time (AOT) Compilation in the context 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 ***concurrent*** *and* ***adaptive*** *JIT compilation engine in order to better utilise todays multi-core architectures. Furthermore the new JIT compilation engine should be extended to support AOT compilation to enable even faster simulation speeds.*

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 process is single threaded, so when the main simulation loop detects a "hot" basic block and dispatches it to the JIT compiler, it has to wait for JIT compilation to finish until the simulation can continue.

The main aim of this project is to decouple the main simulation loop from the JIT compilation engine by introducing **concurrency** in order to achieve the following:

- Continue main simulation loop in interpretive mode while a "hot" basic block is being JIT compiled into native code.
- Enable **concurrent** JIT translations of "hot" basic blocks.

To increase the responsiveness of interactive applications that are simulated, the JIT compilation engine should **adapt** the aggressiveness of the optimisations applied to JIT compiled code depending on the execution frequencies of basic blocks. Thus when a basic block becomes "warm" (i.e. it has been executed a few times) it gets quickly translated (i.e. without optimisations) into native code. If the basic block becomes sufficiently "hot", it will be recompiled using aggressive and thus more time consuming optimisations.

In order to further exploit the processing power of multi-core architectures and increase simulation speed, the simulator should be extended with the capability of AOT compilation. So while the main simulation loop is interpreting or executing already JIT compiled basic blocks, and the JIT compiler is translating currently "hot" basic blocks into native code, an AOT compilation engine should look for basic blocks that might be executed in the future and compile them ahead-of-time.

Finally a comparison of achieved simulation speeds against the baseline single-threaded JIT compiling simulator using standard benchmarks and custom compute and data intensive applications (i.e. AAC decoding, fractal computations) should be performed. A comparison with another state-of-the-art instruction set simulator with respect to simulation 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).

Supervisor: Björn Franke (`bfranke@inf.ed.ac.uk`)

References

[1] J. Ha, M. R. Haghighat, S. Cong, K. S. McKinley. A Concurrent Trace-based Just-In-Time Compiler for Single-threaded JavaScript. Workshop on Parallel Execution of Sequential Programs on Multicore Architectures (PESPMA'09), Austin, TX, June 2009.

[2] John Aycock. A brief history of just-in-time. ACM Computing Surveys, 35, 2, 2003, pp. 97-113.

[3] M. Fulton and M. Stoodley. Compilation Techniques for Real-Time Java Programs. CGO'07: Proceedings of the International Symposium on Code Generation and Optimization, 2007, pp. 221-231.