JIT Development Tasks

This page contains a list of development tasks for the Harmony JIT compiler. If you are new to the JIT, go on and pick a simple task, but do not forget to let us know about your choice in the mailing list. Harder tasks are here to show main strategies of JIT development. Feel free to take them too.

For Beginners

1. Improve command-line parameters handling

Problem: A lot of Jitrino.OPT algorithms use "magic numbers" or hardcoded parameters. See the example of the code with "magic numbers" in the `vm/jitrino/src/dynopt/StaticProfiler.cpp` file.

Task: Refactor code to make it accept parameters using the PMF framework.

2. Improve IR logging code

Problem: Jitrino.OPT uses two intermediate representations (IR) to compile methods: High-level IR and Low-level IR. These IRs use the same base package for Control Flow Graph, Nodes, Edges and Insts representation. However, due to historical reasons, the IR logging frameworks are different.

Task: Refactor and consolidate IR printing code for High and Low level IR representations.

Front End

3. Java bytecode translator refactoring

Problem: interaction between `ByteCodeParser`, `JavaLabelPrepass`, `JavaByteCodeTranslator` classes is very complicated and error-prone.

Task: Re-factor Java bytecode translator in the Jitrino.OPT to make the code cleaner and simplify the data structures used.

4. Clean-up IRBuilder to get rid of simpligier and HVN

Problem: IRBuilder has simplifier and hvn embedded which makes it difficult to use IRBuilder outside of translator.

Task: Re-factor IRBuilder to make it independent from simplifier and HVN. Add "simplify,hvn" optimization passes after translator in the pipeline.

Performance-related improvements in Jitrino.OPT

5. BBP

Task: Reduce overhead from Back Branch Polling: remove BBP from finite loops or reduce overhead, if loop finiteness is undetermined.

New optimizations in Jitrino high-level optimizer

6. Escape Analysis based on-stack allocation

Task: Implement EA-based on-stack allocation

7. Lock coalescing

Task: Implement Lock Coalescing optimization in Jitrino.OPT

Improvements in Jitrino.JET

8. Bytecode-based edge profiling in Jitrino.JET

Task: Bytecode-based edge profile supported by Jitrino.OPT compiler. After implementing on Jitrino.JET JIT may use JET-to-OPT recompilation schemes and employ edge profile in dynamic optimizations in OPT.

Code Generator (CG) Optimizations

9. IPF enabling

Problem: Jitrino.opt contains an IPF code generator, but the rest of the system has only experimental support this platform. For example, on IPF high-level optimizer is disabled. Also, the Jitrino.OPT code generator needs more platform-specific optimizations and tuning.

Task: just do it

10. Branch optimization in IA32/Intel 64 CG

Problem: Analysis of the code generated by Jitrino on IA32 shows that many unnecessary branches could be eliminated.

Task: contact us for live examples and optimize them one by one

11. Register allocation improvements and tuning

Problem: Currently there are 2 global register allocators in Jitrino: bin-packing and color graph.

Task: Improve the register allocation in the following directions:

- Profile-guided live-range splitting in the register allocators.
- Tuning and enhancing the color graph RA.
- Implementation of new register allocation schemes

12. Calling conventions

Problem: Currently the IA-32 CG uses calling conventions that pass all parameters on stack. Also, in both IA-32 and Intel64 CGs all used calling conventions require returning FP values in x87 register stack and do not support callee-saved SSE registers while all FP arithmetic is done using SSE /SSE2.

Task: Improve aggressive inlining, which reduces the overhead performance, in the following directions:

- Passing arguments in GP and SSE registers
- SSE-friendly calling conventions.
- Pluggable calling conventions.

13. Short immediates in generated code

Problem: Currently, all the generated code with immediate constants uses 32 bit consts. Shrinking them for some (normally for ALU and conditional branches) instructions will result in shorter and a bit faster code. (though the exact performance boost is not expected to be significant).

Task: Generate short 8bits immediate constants where possible.

Links

Jitrino chapter from the DRLVM developers guide

The mailing list for Harmony developers. Feel free to ask here.