Inlining

Performance techniques used in the Hotspot JVM

Methods are often inlined, widening the compiler's "horizon" of optimization.

TO DO: Pull in content from the following sources:

- http://mail.openjdk.java.net/pipermail/hotspot-compiler-dev/2008-April/000134.html
- http://forum.java.sun.com/thread.jspa?threadID=790229
- http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6316156
- http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6565458
- http://blogs.oracle.com/watt/resource/jvm-options-list.html

Heuristics

- *TO DO...*
- *Source code:* http://hg.openjdk.java.net/jdk7/hotspot/hotspot/file/tip/src/share/vm/opto/bytecodeInfo.cpp

Phasing

Currently, inlining decisions are performed eagerly (in both C1 and C2), as soon as the parser reaches a call site the first time.

This makes the inlining heuristics less robust, since they must estimate the "heat" of each call site without comparison with call sites not yet encountered. Cold sites are kept out of line, while hot sites are inlined, and recursively parsed immediately. A good project would be to support warm sites (neither hot nor cold), which would be macro-expanded from a priority queue until the estimated frequency drops enough, or the compilation task gets too large.

Profiling

Hot methods are more likely to be inlined... *(Say on...)*

Manual advice to the inliner

There are several ways to help the inliner make the decision you want it to:

- Warm up your code properly.
- Segregate fast paths from slow paths, keeping hot methods small (35 bytecodes or less) with error processing and slow paths in separate methods.
- Request inlining from the compiler oracle. *(Need a ref.)*
- *Future:* Find better heuristics and code them into the JVM.
- *Future:* Add an `@Inline` annotation to the JVM. (Needs hacky POC experimentation first.)
- *Future:* Extend the oracle to include profiling information collected from earlier runs, or generated by an offline configuration tool.