The Java™ HotSpot™ Performance Engine - Consequences

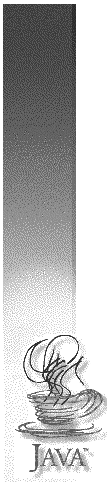
Robert Griesemer
Staff Engineer
Sun Microsystems, Inc.

Talk Overview

- Java Performance
- Software/Hardware Improvements
- Benchmarking
- Some Advice
- Conclusions

© Sun Microsystems, Inc., 1999

Java Performance Today



- Java is getting closer to C++ performance
  - better for some applications
- high-performance Java efforts under way
  - the competition is on!
- “client-side” Java to come
  - shorter startup times
  - more responsive

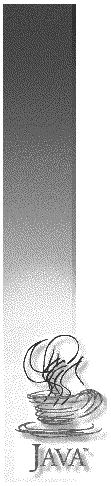
© Sun Microsystems, Inc., 1999

Java Performance Tomorrow

- performance will continue to improve
  - dynamic compilers employ standard optimization techniques
- JITs will be competitive with static compilers for most applications
- next challenge: high-end SMP performance
  - general scalability

© Sun Microsystems, Inc., 1999

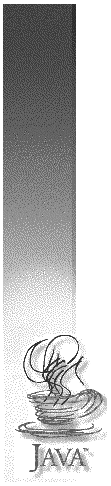
Could Software Help?



- optimizations on the bytecode level
  - javac
  - “source-level” debugging vs. faster execution
- more efficient class file formats
  - encoding for rapid decoding
  - bytecodes optimized for interpretation
  - more program structure information

© Sun Microsystems, Inc., 1999

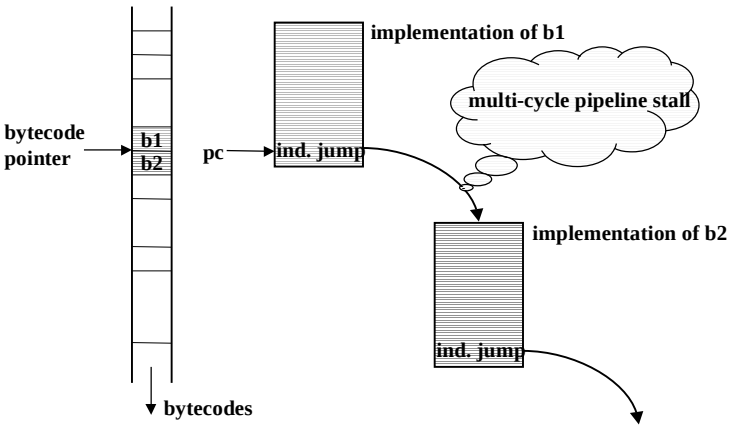
Can Hardware Help?



- standard RISC is just fine, thank you
  - don’t penalize legacy code
- large caches, especially I-cache
  - #1 performance booster
- cheap and selective I-cache flushing
- interpreters could use more support
  - bytecode dispatch is #1 performance killer

© Sun Microsystems, Inc., 1999

Dispatch in Interpreters



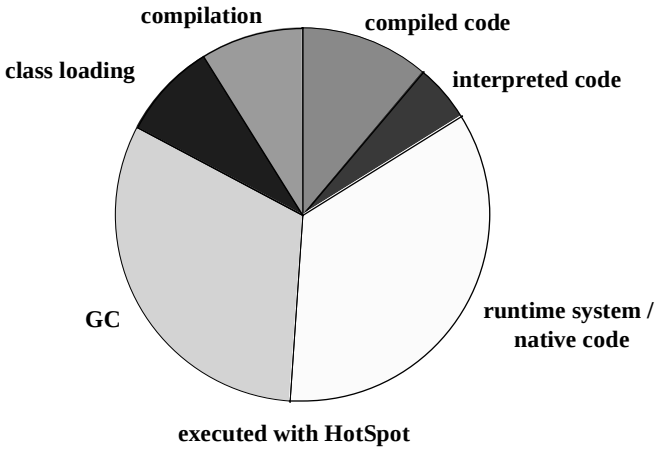
© Sun Microsystems, Inc., 1999

The Limit: Amdahl's Law

- Java optimizations end where Java ends
  - native code
  - garbage collection
  - class loading
- consequences for benchmarks
  - simple benchmarks show unrealistic behavior
  - use real applications!

© Sun Microsystems, Inc., 1999

Example: javacup



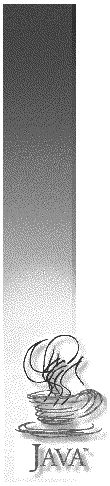
© Sun Microsystems, Inc., 1999

Say No to Micro-Benchmarks!

- small, artificial, C-like micro-benchmarks
- do not reflect real Java applications
  - almost no calls, no dispatch, no allocation, no synchronization, no runtime system calls
- easy target for compiler tricks
- better: e.g., SPEC JVMClient98

© Sun Microsystems, Inc., 1999

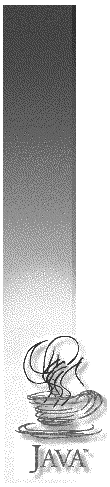
Myth: Small Functions are Expensive



- truth: small functions don't cost!
  - big methods are hard to maintain
  - small methods are inlined
  - good factoring doesn't cost

© Sun Microsystems, Inc., 1999

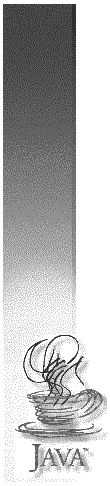
Myth: object creation is expensive



- truth: object allocation is very fast
  - freelists are error-prone
  - freelists may slow down the system
  - GC time proportional to # of live objects
- exception: huge objects
  - may be necessary to reuse them

© Sun Microsystems, Inc., 1999

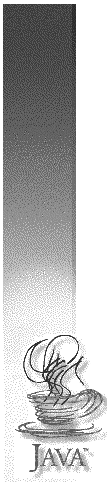
Myth: virtual calls are slow



- truth: inline caches & CHA make them fast
  - no need for static or final just to make system faster
  - CHA often allows inlining

© Sun Microsystems, Inc., 1999

Some Advice



- avoid premature optimization!!!
- apply object-oriented programming style
  - factor code
  - encapsulate data
  - use accessor functions
- profile your application

© Sun Microsystems, Inc., 1999

Conclusions

- Java does not need heavy architectural support
- abstraction becomes free, soon
  - optimizations across components
  - encapsulation without regrets
- uncompromised OO coding style possible
- higher productivity

© Sun Microsystems, Inc., 1999