ByteCode Specialization (BCS): A Run-time Specialization Technique in JVM Bytecode

Hidehiko Masuhara

Dept. of Graphics and Computer Science

Graduate School of Arts and Sciences

University of Tokyo *masuhara@acm.org*

Overview: ***ByteCode Specialization***

- BCS is a ***run-time program specialization*** technique for Java bytecode programs
  - ***dynamically*** optimizes programs using run-time values
  - may generate better code than traditional run-time specialization techniques by ***exploiting JIT compilers***
  - 100% Pure Java

what is ***Program Specialization***?

a technique to generate an ***efficient program***
from a ***generic program***
and ***some parameters*** (e.g., partial evaluation)

example:

```
int power(int x, int n)
```
{ computes x^n }

n=3

specialization

```
int power_3(int x)
{ return x*x*x; }
```

optimized

x=9

efficient execution

729

advantages of ***Run-Time*** Specialization

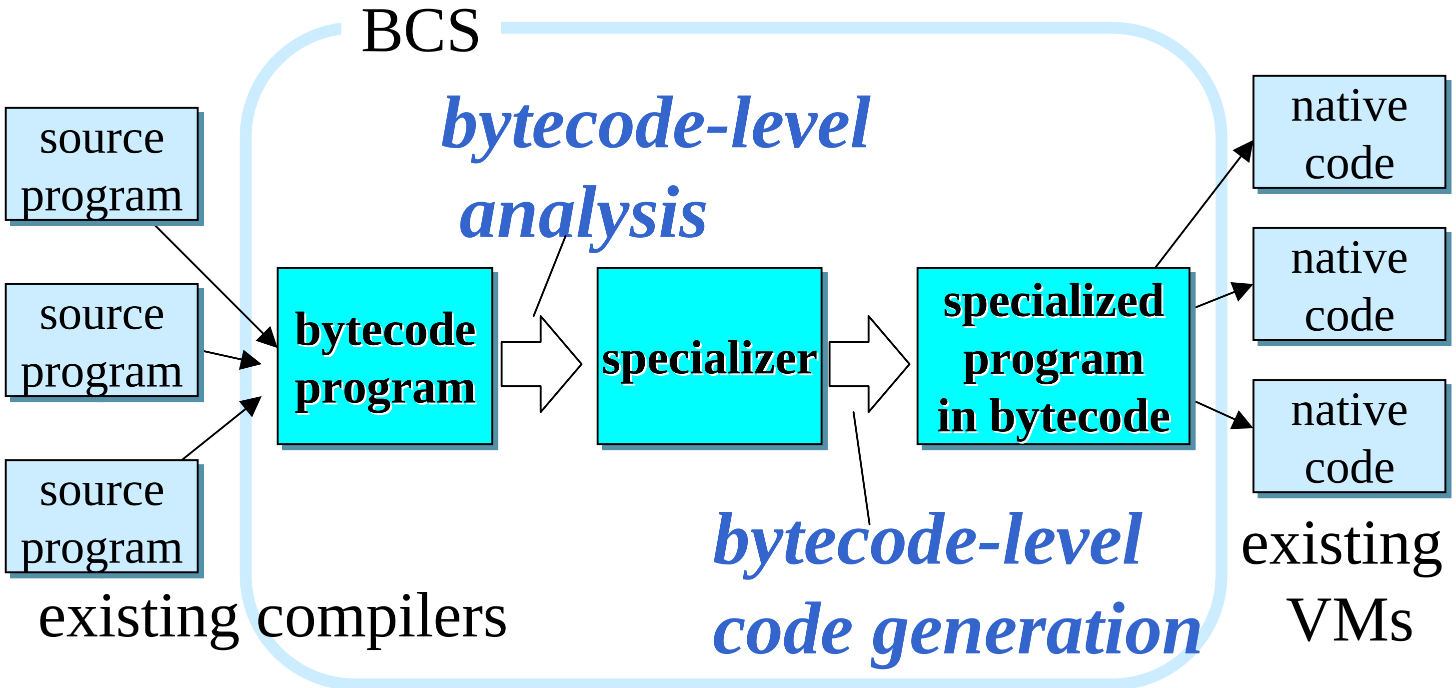
- can optimize programs by ***using run-time values***
  - ***eliminate computation*** that depends on known parameters
  - performs high-level optimizations
    e.g., ***method inlining*** whose target class can only be determined at run-time
    e.g., ***loop unrolling***, ***constant folding***, ...

existing specialization techniques

- Partial evaluation: [Futamura71, Jones93, etc.]
  - source-to-source transformation
    i.e., needs ***compilation after specialization***
  - interpretive (= ***slow***) specialization
- Run-time specialization [Engler94,Lee&Leone94, Consel96,etc.]
  - code generation at binary level
    i.e., ***very fast*** specialization process
  - ***less optimized*** specialized code
  - highly dependent on the target architecture

ByteCode Specialization (BCS)

a run-time specialization technique
on Java Virtual Machine bytecode



Binding-Time Analysis in BCS

- directly analyzes bytecode programs (no source program is required)
- uses a ***type system*** for basic constraints + ***flow analysis*** for side-effects

states of stack ***before*** & ***after*** the instruction

a typing rule:

$$\frac{P[pc] = \mathbf{iadd},\ T_{pc} = \beta \cdot \gamma \cdot \sigma,\ T_{pc+1} = \alpha \cdot \sigma,\quad \beta \le \alpha,\ \gamma \le \alpha,\ \alpha = B[pc]}{A, R, B, F, T, pc \triangleright P}$$

constraints

Rules for Binding-Time Analysis

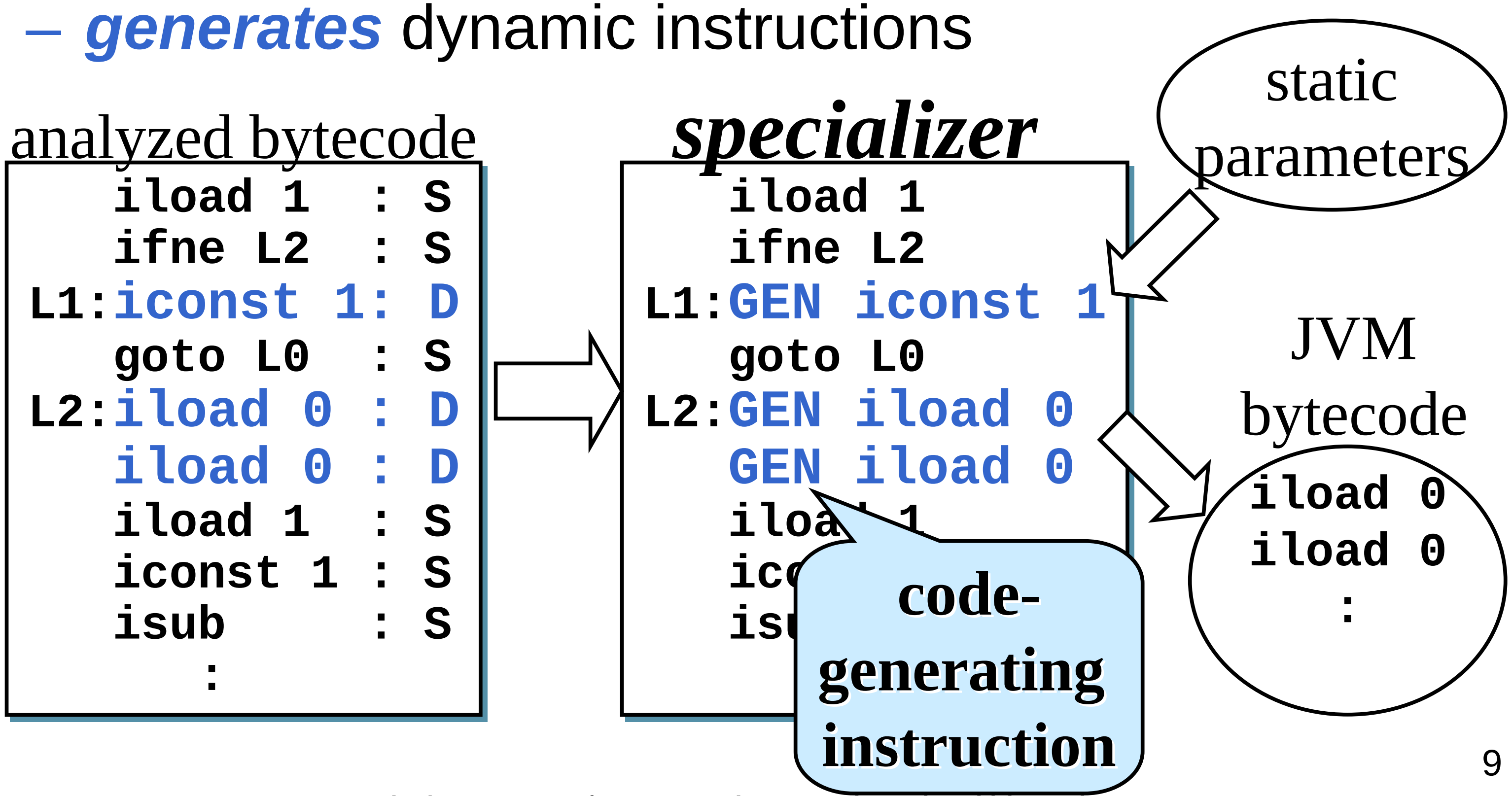
(extension of Stata&Abadi's [Stata&Abadi98])

$$\frac{\begin{array}{c} P[pc] = \texttt{iconst}\ n \\ F_{pc} \subseteq F_{pc+1} \\ T_{pc+1} = \alpha \cdot T_{pc} \\ B[pc] \leq \alpha \end{array}}{A, R, B, F, T, pc \vdash P}
\quad
\frac{\begin{array}{c} P[pc] = \texttt{iadd} \\ F_{pc} \subseteq F_{pc+1} \\ T_{pc} = \alpha \cdot B[pc] \cdot \sigma \\ T_{pc+1} = \beta \cdot \sigma \\ \alpha \leq B[pc] \leq \beta \end{array}}{A, R, B, F, T, pc \vdash P}
\quad
\frac{\begin{array}{c} P[pc] = \texttt{iload}\ x \\ F_{pc} \subseteq F_{pc+1} \\ T_{pc+1} = \alpha \cdot T_{pc} \\ F_{pc}[x] = B[pc] \leq \alpha \end{array}}{A, R, B, F, T, pc \vdash P}
\quad
\frac{\begin{array}{c} P[pc] = \texttt{istore}\ x \\ F_{pc}[x \mapsto B[pc]] \subseteq F_{pc+1} \\ \alpha \cdot T_{pc+1} = T_{pc} \\ \alpha \leq B[pc] \end{array}}{A, R, B, F, T, pc \vdash P}$$

$$\frac{\begin{array}{c} P[pc] = \texttt{ifne}\ L \\ F_{pc} \subseteq F_{pc+1}, \quad F_{pc} \subseteq F_L \\ \alpha \cdot T_L = \alpha \cdot T_{pc+1} = T_{pc} \\ \alpha \leq B[pc] \end{array}}{A, R, B, F, T, pc \vdash P}
\quad
\frac{\begin{array}{c} P[pc] = \texttt{invoke}\ m\ n \\ F_{pc} \subseteq F_{pc+1} \\ T_{pc} = A_m[n-1] \cdots A_m[0] \cdot \sigma \\ T_{pc+1} = \alpha \cdot \sigma \\ R[m] \leq \alpha \end{array}}{A, R, B, F, T, pc \vdash P}
\quad
\frac{\begin{array}{c} P[pc] = \texttt{ireturn} \\ T_{pc} = \alpha \cdot \epsilon \\ \alpha \leq R[m] \\ pc = \langle m, i \rangle \end{array}}{A, R, B, F, T, pc \vdash P}$$

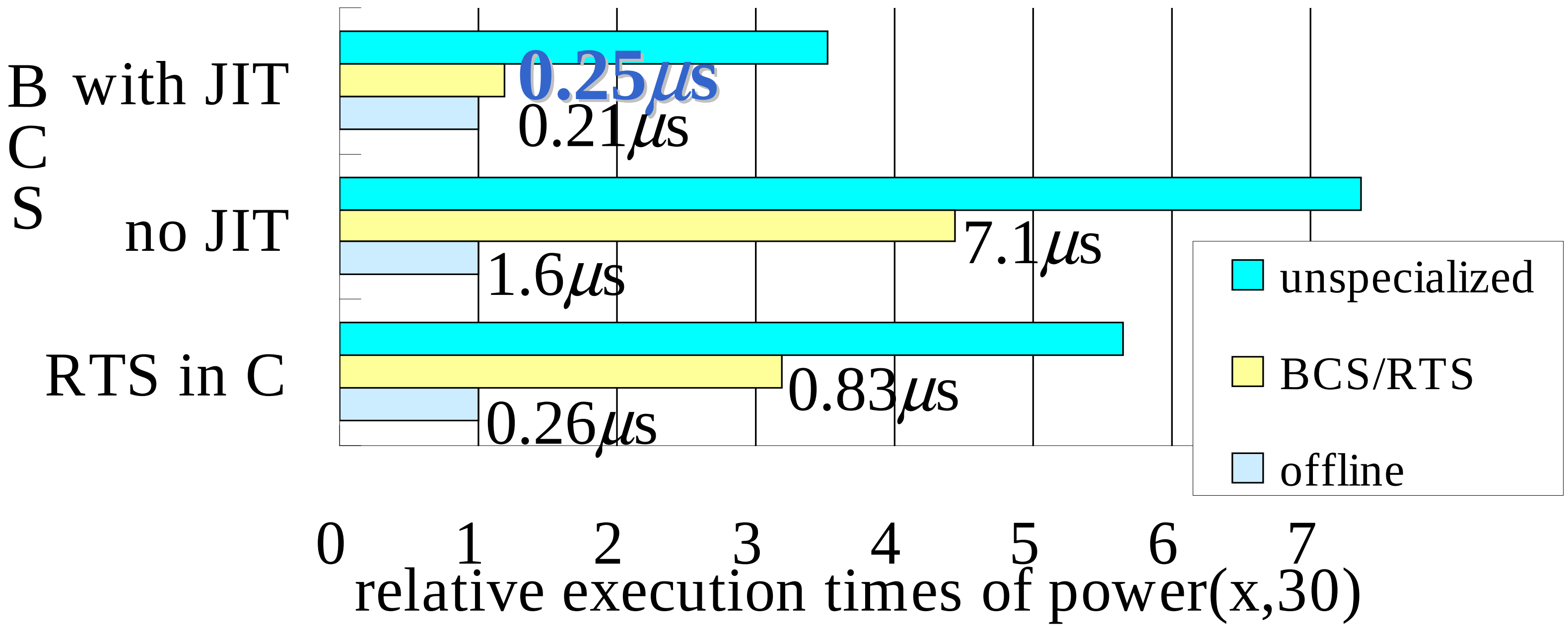
construction of a ***specializer***

- specializer is a program that
  - ***performs*** static instructions
  - ***generates*** dynamic instructions



Performance: ***specialized code***

- 3x faster than unspecialized
- 3x faster than traditional RTS
- close to offline specialization

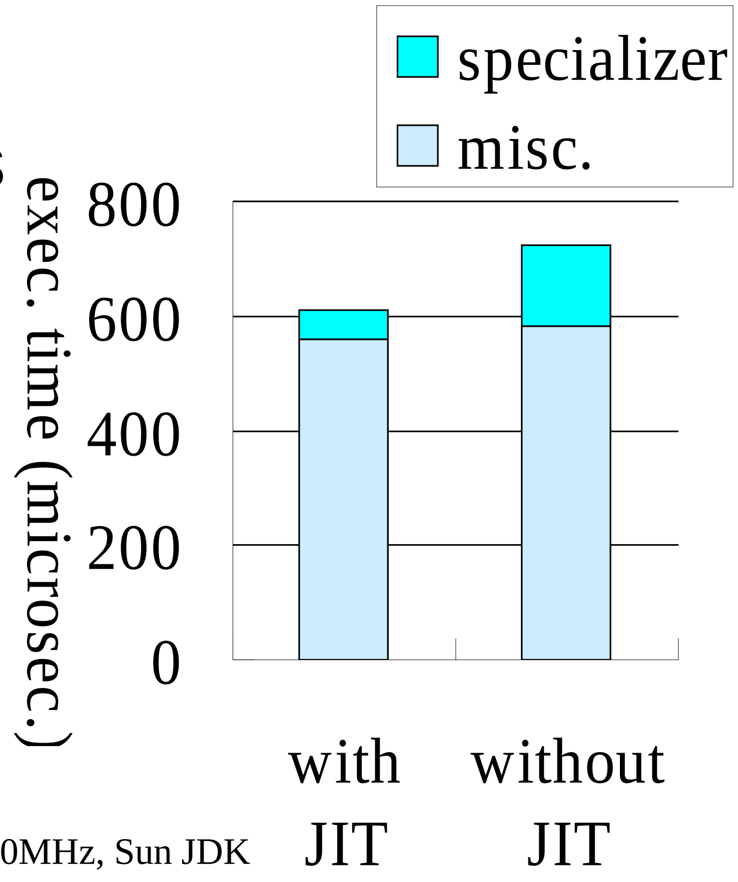


*does not include specialization time

Pentium II 300MHz, Sun JDK 1.1.7+Symantec JIT, GCC 2.7.2

Performance: ***specialization speed***

- ***much faster*** than traditional compilers
- overheads in JVM class loader?



Pentium II 300MHz, Sun JDK 1.1.7+Symantec JIT, GCC 2.7.2

Future work

- extend the system to
  the full version of JVM
- better integration with existing programs (*e.g.*, specialization classes [Volanschi97])

- for more info:

  "Run-Time Program Specialization in Java Bytecode" in Proc. of SPA'99. (available from http://www.graco.c.u-tokyo.ac.jp/~masuhara/BCS)