



**HAL**  
open science

## Evaluation of the runtime performance of control flow structures for dynamic dispatch in Java

Olivier Zendra, Karel Driesen, Feng Qian, Laurie Hendren

► **To cite this version:**

Olivier Zendra, Karel Driesen, Feng Qian, Laurie Hendren. Evaluation of the runtime performance of control flow structures for dynamic dispatch in Java. Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA), 2001, Tampa, United States. inria-00563339

**HAL Id: inria-00563339**

**<https://inria.hal.science/inria-00563339>**

Submitted on 4 Feb 2011

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Olivier Zendra & Karel Driesen (INRIA – McGill University, ACL group) – Feng Qian & Laurie Hendren (McGill University, Sable group)  
 {ozendra,karel,fqian,lhendren}@cs.mcgill.ca

## Motivation:

- Frequent dynamic dispatch in Java programs
- Efficient implementation crucial
- Which bytecode control structure is the most efficient to implement dynamic dispatch ?

## Methodology:

- A benchmark suite to time control structures for dynamic dispatch in Java under varying conditions:
  - Different control structures
  - Different numbers of possible receiver types (static)
  - Different patterns of runtime receiver types (dynamic)
  - Different execution environments

Java benchmarks are generated for each combination of control structure and number of possible types  
 ↗ 220+ variants

Type ID pattern files are generated, representing the order in which types occur at runtime  
 ↗ Control & repeatability  
 ↗ 60+ variants

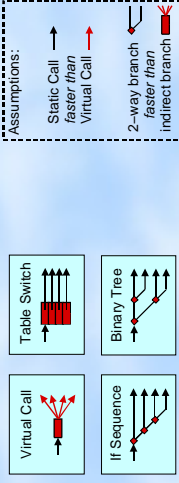
We time the execution of each benchmark on all patterns that apply to it, on various execution environments:

- Java Virtual Machines:
  - SUN HotSpot Server [HSS]
  - SUN HotSpot Client [HSC]
  - IBM JIT
- Hardware platforms:
  - Intel Pentium III [P-3]
  - SUN UltraSparc III [U-3]
  - Intel Celeron
  - (AMD Athlon)

Bench × pattern × JVM × platform = 21,000 data points

## Control structures:

We generate different control structures for polymorphic calls:

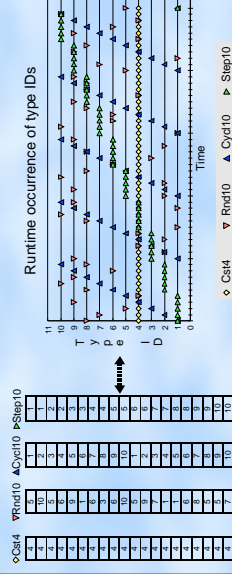


For comparison we also run benchmarks without any call:  
 No Call →

## Type ID Patterns:

Frequency	Very common	Common	Uncommon	Average High	High	Cyclic	Stepped
Constant	High	High	High	High	High	High	High
Non-Constant	High	High	High	High	High	High	High
Very common	High	High	High	High	High	High	High
Common	High	High	High	High	High	High	High
Uncommon	High	High	High	High	High	High	High
Average High	High	High	High	High	High	High	High
High	High	High	High	High	High	High	High
Cyclic	High	High	High	High	High	High	High
Stepped	High	High	High	High	High	High	High

Implementation: x,000 iterations over a 10Kint array → x0,000,000 calls

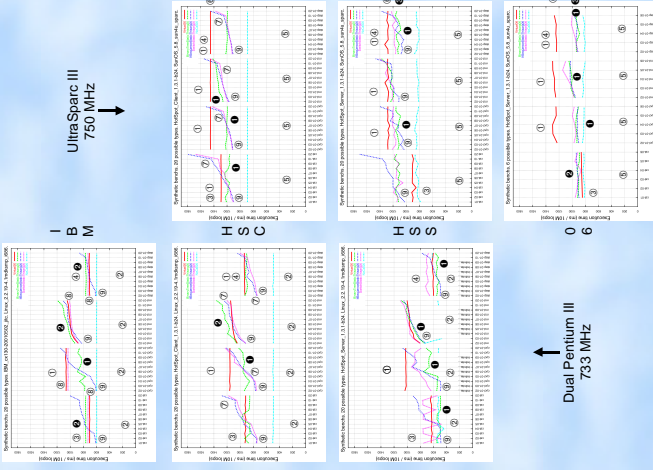


## Some results:

- Virtual calls tend to be more expensive than other structures
- Virtual calls on P3 are sensitive to execution patterns
- Virtual calls on constant patterns tend to be efficient on P3 and U3
- Virtual calls with stepped patterns behave well on P3 but not on U3
- Virtual calls on U3 are insensitive to execution patterns, except for the constant pattern
- Virtual calls on U3 tend to be costly
- Switches on HSC behave like if sequences
- Switches on IBM behave like virtual calls
- If sequences are the best when only the first few ifs are exercised, except in some cases for HSS on U3

- Binary trees are significantly faster than virtual calls in most cases
- When binary trees are slower than virtual calls, the difference generally remains small

- All control structures, except virtual calls, speed up when the number of possible types is small



## Conclusion:

- Performance of dynamic dispatch varies greatly across VMs and execution patterns
- Dynamic dispatch in current VMs can be significantly improved at compile time
- Much better implementations are possible for small numbers of possible receiver types, if sequences for up to ≈4, binary tree dispatch up to ≈8