Intentional Benchmarking of Dynamic Languages

By Maisha Labiba and Dave Mason
Toronto Metropolitan University

Introduction

Benchmarking is often treated as a very *ad hoc* activity. This talk approaches benchmarking from a more principled perspective, laying out the goals of benchmarking, then looking for benchmarks to fulfil the goals.

We are selecting and developing benchmarks that will expose various characteristics of dynamic language implementations.

Many benchmark suites have been developed over the years but only Marr *et al* have discussed in detail how they selected benchmarks in similar terms to ours.

Contents

- History
- Approach
- Benchmarking Questions/Targets
- Choosing Benchmarks
- Future Work

History

- ad hoc
  - used by compiler writers
  - used by customers
- micro-benchmarks
- SPEC wars - C compilers and hardware
- De Capo - Java progress
- Are We Fast Yet?

Our Approach

- benchmarks must be repeatable
  - every run produces exactly the same data and control flow
  - mustn’t use the system random number generator
- benchmarks must be comparable
  - execution on different languages or implementation of languages have the same flow
    - difficult for wildly different paradigms
  - mustn’t use a foreign-function-interfaces (unless that is the target of the benchmark)
  - mustn’t use other special libraries
- benchmarks that exercise particular targets (synthetic benchmarks)
- as well as others that are more holistic (natural benchmarks)
  - but that remain small enough to be implemented easily in various languages
  - or have public repos of multiple language implementations
- identifiable characteristics
  - so users could use for decisions

Benchmarking Questions/Targets

As we want to evaluate the quality of language implementations, we focus on the following targets:

1. basic arithmetic operations
2. polymorphism, class hierarchies, and dynamic method dispatching,
3. closures/lambdas, and exception handling
4. deep recursion,
5. parallel/multi-processing,
6. heavy array processing
7. memory management/garbage collection of large objects,
8. memory management/garbage collection of many small objects.

Choosing Benchmarks

- N-sieve
  - integer arithmetic
  - simple array operations
- fibonacci and fib-memo
  - heavy recursion and cached
- Mandlebrot
  - large number of *independent* calculations - highly parallelizable/multi-processor
- N-body simulation
  - large number of *dependent* calculations - parallelizable with heavy interactions
- matrix inversion
  - array and floating-point
- data-frame
  - large array; heavy GC demand
- splay trees
  - many small object; different GC demand

Results

- oh…. you were expecting results
- see next slide

Future Work

- run the benchmarks on variety of Smalltalk implementations
  - Zag, Pharo, Bee, VAST, VW
- run the benchmarks on variety of dynamic languages
  - Python, Ruby, Javat
- run the benchmarks on variety of static languages
  - C, C++, Zig, Rust
- publicize results
- fill gaps