



The HotPy Virtual Machine



University
of Glasgow

Mark Shannon

University of Glasgow



HotPy



- Recursive acronym:
 - **H**otPy
 - **O**ptimising
 - **T**ranslating
 - **P**ython
- **C**ompiling
 - **O**ptimising
 - **P**ython



HotPy Features

- Fast
 - 2-3 times as fast as CPython3.1
- Fast start-up
 - All builtins are included in executable
- Concurrent
 - No Global Interpreter Lock
 - Can use multiple cores/processors for greater speedup



HotPy Features(2)

- Fast
- Fast start-up
- Concurrent (no Global Interpreter Lock)
- **Incomplete**
 - No interactive interpreter: need a new parser
- **Untested**
- **Batteries not included**



HotPy design

- Build using the Glasgow Virtual Machine Toolkit
- Designed to *optimisable*, rather than optimised
- Internal design as well-factored as possible
 - Eases optimisation, so improves performance
 - Example:
 - All binary operators share a common bytecode and dispatching mechanism



The GVMT

- Glasgow Virtual Machine Toolkit
- Provides an abstract stack machine layer
 - Garbage collection
 - Exceptions
 - Efficient translation of stack code to underlying machine code
- Automatic generation of interpreter *and* compiler from same specification
- HotPy is built with the GVMT



Internal design

- Object layout:
 - All objects have a number of slots predefined.
 - Reduces locking and allows attribute optimisations
- No slots in classes for standard methods
 - All methods reside in class dictionary
 - Consistent lookup for *all* methods
- Operators are implemented as multimethods
 - Fast lookup of common operations



Quick aside: Multimethods

- Dispatch on the type (class) of all of their parameters.
- `add(x, y)`:
 - dispatches on tuple `(x.__class__, y.__class__)`
- In general a n-ary multimethod is a mapping of n-tuples of classes to behaviours (functions).
- HotPy binary operators use this mapping only for exact matches
 - then fall back on standard Python lookup
 - ensures that HotPy has the same semantics



Garbage Collection



University
of Glasgow

- HotPy uses a copying collector, because:
 - Concurrency is impossible with reference counting
 - Python programs generate a lot of garbage and lots of small objects, which suits copying collectors
- The collector is provided by the GVM T
 - Garbage collectors in GVM T are pluggable
 - Currently only a simple semi-space collector
 - Hope to have a generational and maybe parallel collector soon



Concurrency

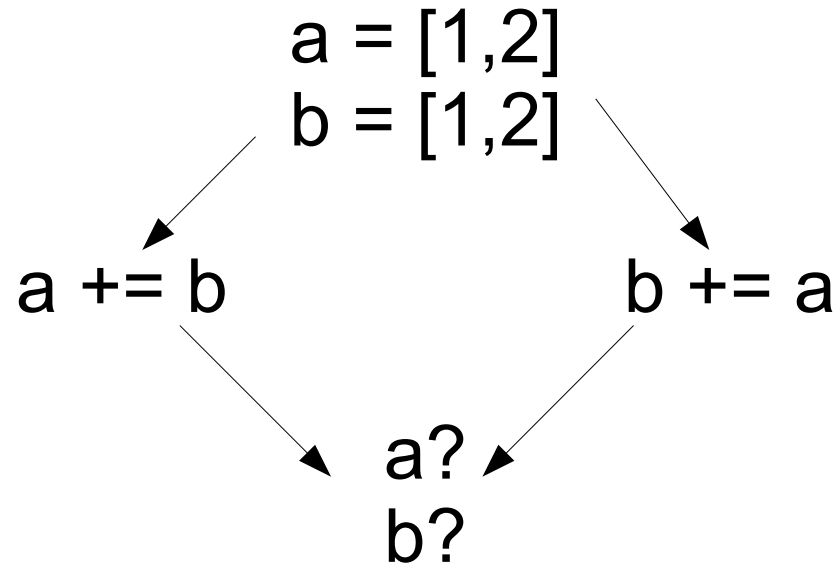


University
of Glasgow

- HotPy has no Global Interpreter Lock
- Finer grained locking is required
 - To maintain the integrity of the virtual machine and core data structures.
- This locking has a cost:
 - Write only locks: ~15% overhead
 - Read/write locks: ~35% overhead
- But concurrency will pay much more than locking costs.



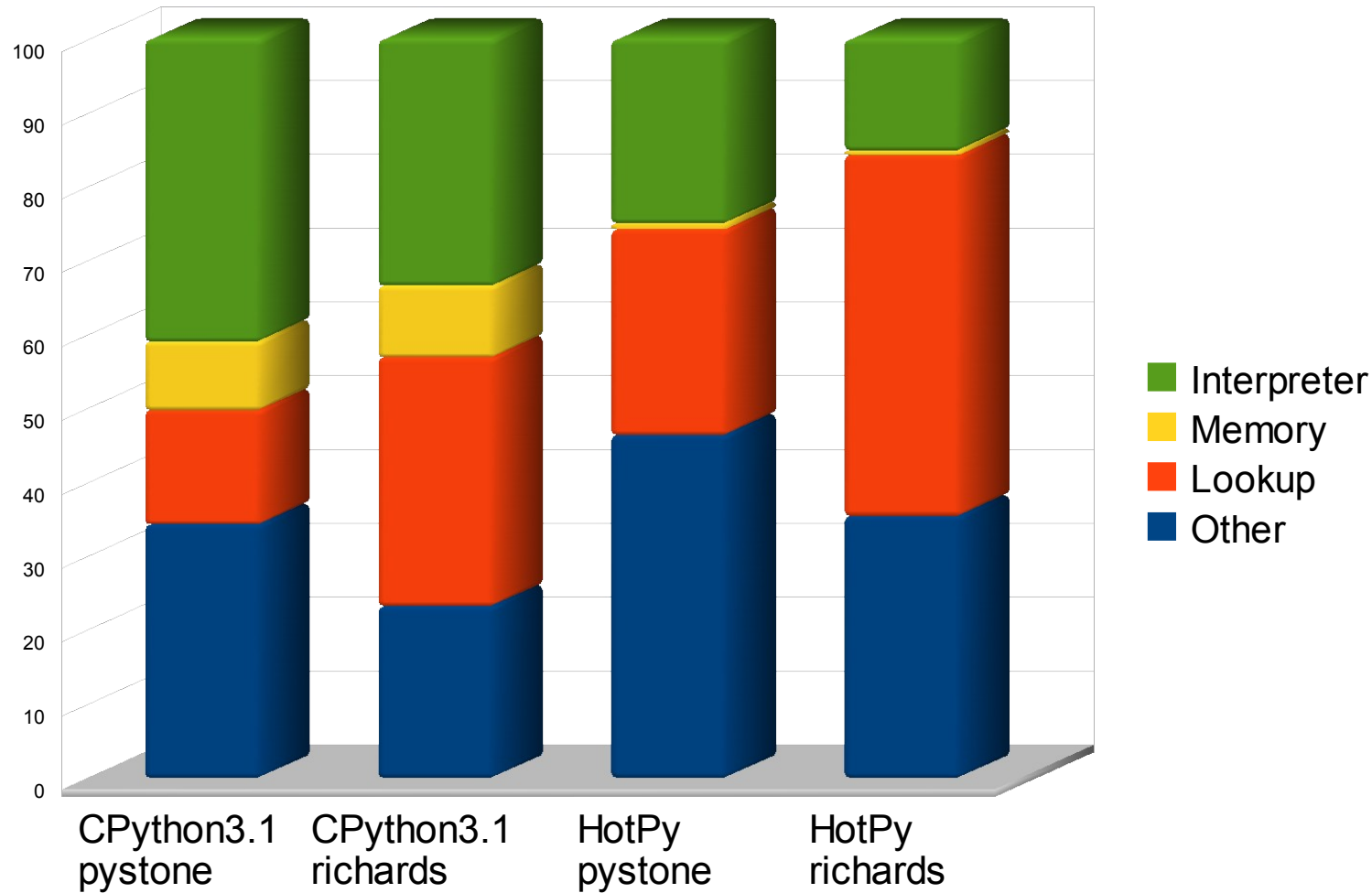
Concurrency(2)



- No sensible result for above code.
- Read locks cannot protect programmer from concurrency issues.
- HotPy uses write only locks.
- Synchronisation is up to the programmer



CPython and HotPy



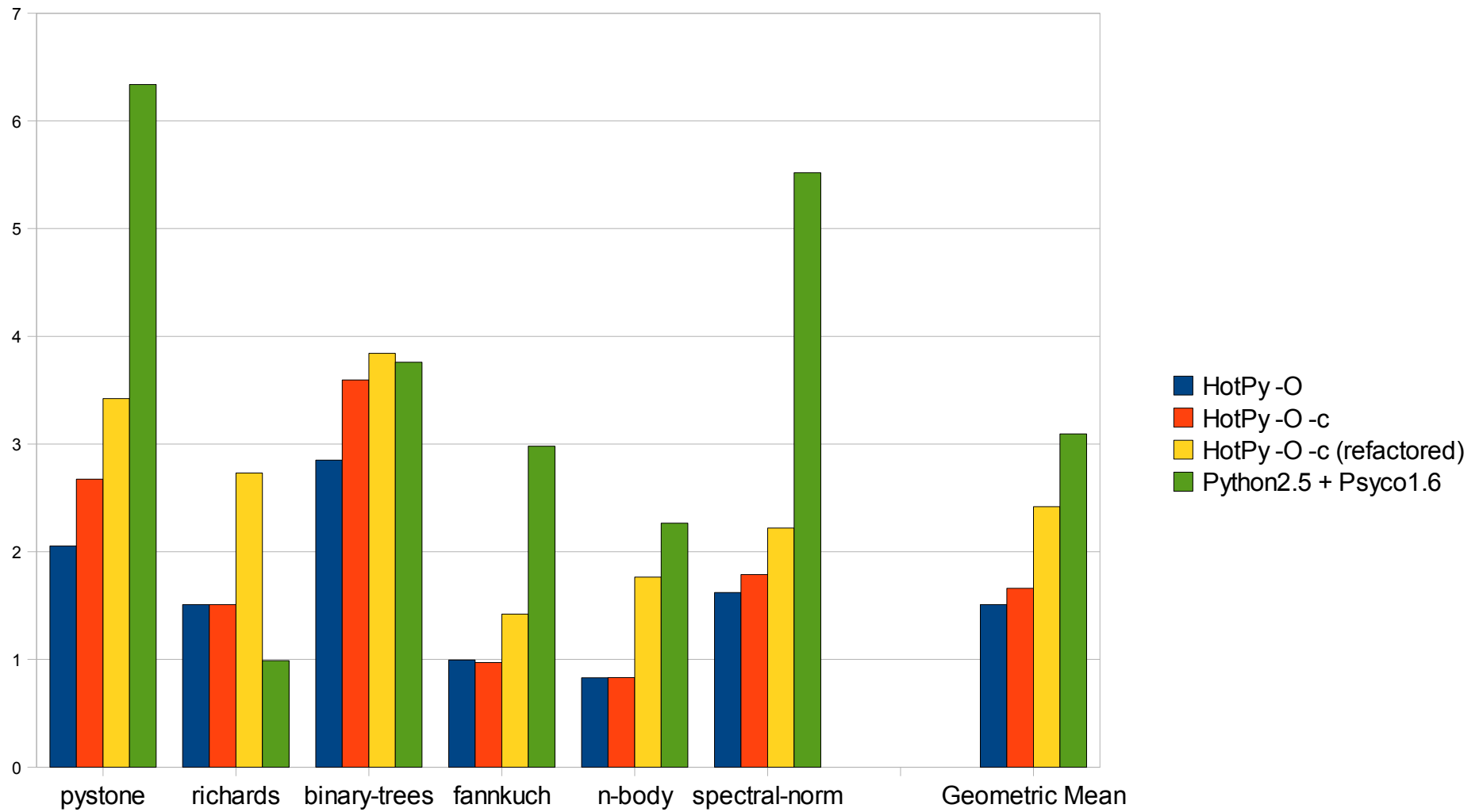


Benchmarks

- Pystone
- Richards
- Four benchmarks from the
Computer Language Benchmarks Game:
 - <http://shootout.alioth.debian.org/u32q/benchmark.php?test=all&lang=python3>

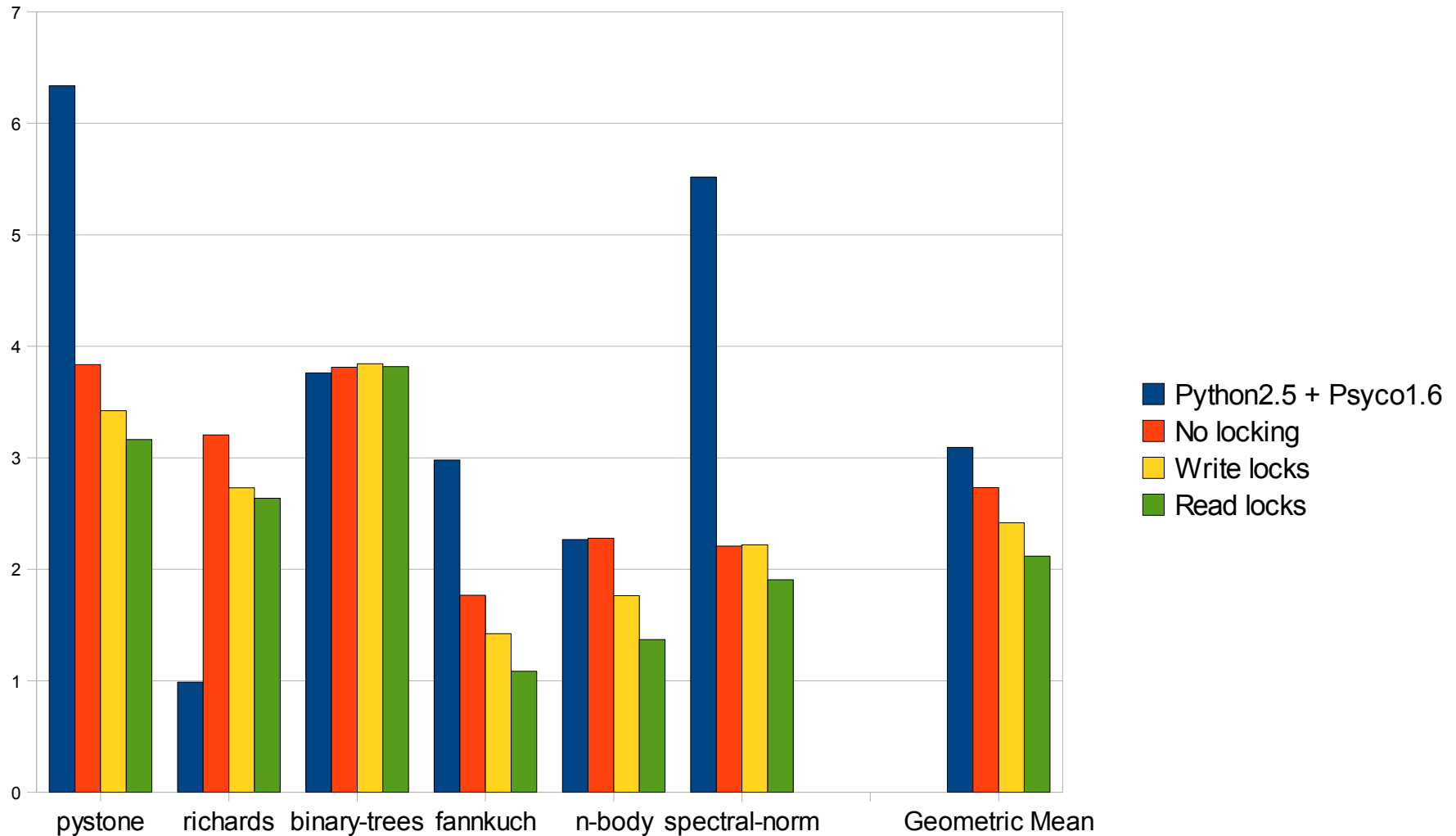


Benchmark results





Effect of locking





Future work



University
of Glasgow

- Add better GC to the GVMT.
 - Will enable testing of concurrency and some concurrent benchmarks
- Implement trace-based optimiser
 - Expect a further 10-20% speedup beyond the refactored speeds.
- Complete core object implementations
- A proper bytecode compiler
- Lots more to do...



Help needed

- If HotPy is ever to be useful then your help is required
- I need to write up my PhD
 - Limits the amount of time I can spend on HotPy
- Money!
 - Fund continuing working on HotPy



Any questions?



University
of Glasgow

- Thanks for listening
- Any questions?
- <http://www.dcs.gla.ac.uk/~marks>
- <http://gvmt.sourceforge.net/>