# Python without the GIL

Armin Rigo
Maciej Fijałkowski

PyCon US 2013

March 15 2013

## Intro

- PyPy is a Python interpreter with stuff
- No general PyPy talk this year, find us around, come to the BoF (tomorrow 2pm)

## This is about...

- This talk is about using multiple cores to achieve better performance
- in Python (or any other existing, non-trivial, non-functional, non-designed-for-this-problem, language)

## Problem

- An existing complex, large program that does stuff
- “stuff” consists of bits that are mostly independent from each other
- ... but not quite

## Problem

- An existing complex, large program that does stuff
- “stuff” consists of bits that are mostly independent from each other
- ... but not quite

## Problem

- We want to parallelize the program to use all these cores
- We have some shared mutable state
- Not too much of it --- otherwise, no chance for parallelism
- But still some

## Problem

- We want to parallelize the program to use all these cores
- We have some shared mutable state
- Not too much of it --- otherwise, no chance for parallelism
- But still some

## Classic solutions

Bare-metal multi-threading:

- large shared state
- needs careful usage of locks
- mostly hindered by the GIL in CPython (but not in Jython or IronPython)

## Classic solutions

Multi-processing:

- no shared mutable state at all
- copying, keeping in sync is your problem
- serializing and deserializing is expensive and hard
- memory usage is often multiplied (unless you're lucky with `fork`, but not on Python)

## Classic solutions

A range of intermediate solutions:

- MPI: message passing, with limited shared state
- etc.: tons of experiments that never caught on in the mainstream

## Classic solutions

The typical solution for web servers:

- run independent processes
- share data only via the database
- the database itself handles concurrency with transactions

## Demo

- `pypy-stm`
- internally based on “transactions” (STM, HTM)

## Demo

- demo:
  - multithreading.py
  - multithreading2.py
  - message_passing.py
  - transactions2.py

## Status of the implementation

- mostly works
- major GC collections missing (leaks slowly)
- JIT integration is not done
- tons of optimizations possible

# How do I use it?

- just like with the GIL
- `__pypy__.thread.atomic`
- `with atomic: print "hello", username`
- the illusion of serialization

## STM - low level

- STM = Software Transactional Memory
- Basic idea: each thread runs in parallel, but everything it does is in a series of “transactions”
- A transaction keeps all changes to pre-existing memory “local”
- The changes are made visible only when the transaction “commits”

## STM - low level (2)

- The transaction will “abort” if a conflict is detected, and it will be transparently retried
- Non-reversible operations like I/O turn the transaction “inevitable” and stop progress in the other threads
- `__pypy__.thread.last_abort_info()` -> traceback-like information

## Alternative - HTM

- Intel Haswell (released soon) has got HTM
- great for the “remove the GIL” part
- not so great for large transactions, at least for now

## Higher level: Threads Are Bad

- based on (and fully compatible with) threads
  - existing multithreaded programs work
- but opens up unexpected alternatives

## Higher level: Atomic

- we can run multiple threads but at the same time use `atomic`
- with the GIL-based idea of `atomic` it wouldn't make sense
    - multiple threads
    - but they're all using `atomic`
    - i.e. only one at a time will ever run
    - ...except no :-)

## Transactions

- `transaction.py`: example of wrapper hiding threads
- illusion of serial execution: can “sanely” reason about algorithms

## Transaction conflicts

- “Conflict tracebacks”
- Might need to debug them --- but they are performance bugs, not correctness bugs
- The idea is that we fix only XX% of the bugs and we are done
- Maybe some new “performance debugger” tools might help too

## We're not the only ones

TigerQuoll, 1st March: same idea with JavaScript (for servers)

Various models possible:

- events dispatchers
- futures
- map/reduce, scatter/gather

## Event dispatchers

- twisted, tulip, etc.
- run the event dispatcher in one thread (e.g. the main thread), and schedule the actual events to run on a different thread from a pool
- the events are run with `atomic`, so that they appear to run serially
- does not rely on any change to the user program

## Donate!

- STM is primarily funded by donations
- We got quite far with $22k USD
- Thanks to the PSF and all others!
- We need your help too

## Q&A

- `http://pypy.org`
- `http://bit.ly/pypystm`