Using All These Cores: Transactional Memory under the hood

Introduction

- Armin Rigo, PyPy dev, CPython dev
- Co-author: Remi Meier, ETHZ
- This talk applies to Python or any similar language

Problem

- Most computer’s CPUs today have multiple cores
- How to use them?

Multithread programming

- C, C++, Java, .NET, ...
- Jython, IronPython

CPython, PyPy

- No story so far
- Alternatives for various cases
- Some fine and some horrible

The GIL

- Global Interpreter Lock
- “Each bytecode is executed atomically”

Transactional Memory

- Recent research (past ~10 years)
- Optimistically runs multiple threads even if they are supposed to be waiting on the same lock
- Usually, high overheads

Expected results

- Runs multiple threads despite a single GIL
- Does not remove the GIL, but solves the original problem anyway

Transactional Memory

- STM: Software Transactional Memory
- HTM: Hardware Transactional Memory
- Hybrids

Status

- STM is still at least 2x slower (on one core)
- HTM: tested in Ruby with Intel Haswell CPUs, not bad but still disappointing (imo)

STM C7

- c7 is our group's research (there were a lot of previous research that failed to give good results)
- Hope: much less than 2x slower for "PyPy-like" usages
- (insert description here)

Atomic sections

- GIL = “each bytecode is atomic”
- One bytecode? Obscure for the regular Python programmer
- Larger atomic sections:

```
with atomic:   ...
```

Larger atomic sections

- New way to synchronize multiple threads
- All `atomic` blocks appear to run serialized
- With STM/HTM, they actually run in parallel as far as possible

No threads?

- Works even if you don't use threads!
- If the Twisted reactor (say) was modified to start a pool of threads, and to run all events in "`with atomic:`"
- ...Then the end result is the same, for any Twisted program

Behind-the-scene threads

- The thread pool added behind the scene lets a STM/HTM-enabled Python run on several cores
- The “`with atomic:`” means that the semantics of the Twisted program didn’t change

Summary (optimistic)

- If you are using Twisted...
- Just wait and your program will run on multiple cores $:-)$

Conflicts

- Actually, your program will likely fail to use multiple cores out of the box
- ...Because of “conflicts”: each event should be “often” independent, but may not be
- Example: incrementing a global counter, or otherwise changing some global object systematically

Some work left for you to do

- You need to figure out where the confícts are
- Maybe using some debugger-like tools that report conflicts
- Then you need (hopefully small) rewrites to avoid them

Some work left for us to do, first

- Additional conflicts come from Twisted itself
- Example: the logging system, which may need to use queues
- This means that some of the core Python data structures (dicts, queues...) may need refactorings too

What is the point?

- The point is that with STM/HTM your program is always *correct* (as much as the single-core version is)
- You need to work in order to fix the most obvious conflicts
- If you don't, it won't be faster than the single-core original

What did we win?

- Regular approach to multithreading: your program is always *fast*
- You need to work in order to fix the bugs (races, deadlocks...)
- You need to find and fix *all* bugs -- as opposed to the STM/HTM version where you only fix *some* issues until it is fast enough

Scope

- Twisted / Eventlet / Stackless / etc.: event-driven programming
- Any program computing something complicated, e.g. over all items in a dictionary, occasionally updating a shared state, etc.
- In general, any CPU-bound program with identifiable sections that have a good chance to be parallelizable: “a good chance” is enough

Conclusion

- Mostly theoretical for now: there is a risk it won't work in practice [1]
- Expect progress in the following months: `http://morepypy.blogspot.com/`

—

[1] I bet it will, eventually :-)