Hacking pypy

Hi everyone. I hope you had fun yesterday, today we'll be jumping a bit into deeper water and hacking PyPy itself. Since this is a big topic, I'll give you a bit of an overview and then guide you through some of the recent work that I've done, so we have somewhat of an understanding of where goes what.

Layers

PyPy is like ogres or onions - it has lots of layers and peeling them sometimes makes you cry. On the plus side, something that on JVM takes years to pass through a committee, can be done in PyPy within a few days. We've done some of the introduction to RPython, now we'll walk through quickly around all the layers.

RPython toolchain

The directory we'll be touching a lot less is RPython. It contains a way to translate things to C, which is mostly stable and does not need touching with rare exceptions and then the interpreter itself which is the topic of today.

flowspace, annotator,
rtyper, translator

Those four parts are translating rpython to C. It's a bit unlikely we'll need to hack most parts of it, but it's important to know where they are in case you need to add a new object layout. They do, correspondingly - flow the graphs to create graphs, annotate the graphs with high level types, lower the types to low-level and generate C.

low level types

Low level types are the direct-to-the-metal way of describing things in RPython. lltype.XXX and rffi.XXX are the primary interfaces that we will be using today, with llmemory.XXX being an extra one if you want to hack on GC.

code1, code2

One of the most important parts that we might not be modifying but we'll be using extensively is rlib. This is RPython standard library as almost all Python stdlib is not RPython. It has a variety of things, including bindings to openssl, unicode/utf8 primitives and tons of other things that seem useful for implementing various interpreters. The most important parts we'll be using are objectmodel, which includes various specializations, we_are_translated, we_are_jitted and such as well as the jit module which contains various hints.

jit

One of the main beasts in RPython is the jit directory - I'll skip over it for now, but it contains all the details of the just in time compiler generator. We can come back to it if we find ourselves with spare time.

memory

This is where our garbage collector lives, currently incremental minimark with nursery, moving objects and write barrier. There are other, older garbage collectors that are present there as well as some bookkeeping code and the GC transformation done on graphs. Unless you hack on the GC, low chances of doing stuff there

pypy subdirectories

Most of pypy is written in RPython with some meta programming written in Python.

interpreter, objspace

The basic stuff is done in the interpreter with specific types done in objspace. The distinction is mostly historical when we had more than one objspace, but it turned out to not be very pluggable :-) objspace contains equivalent of Objects/ directory in CPython while interpreter is Python/ directory.

module

In module directory there are various C-level modules, which are implemented in RPython, like _cffi_backend. Note that some modules are implemented either in pure python or using cffi for simplicity and translation time.

before we start...

One thing that is very different in PyPy than in any other project I've ever worked on. Since the translation times are large, but we control the entire environment, we often rely very heavily on tests and we are very test driven. We often try to describe a general solution that can be used by many interpreters, implement it in RPython with tests and then use it in PyPy and only THEN we translate. This is crucial that some people completely miss - if you just try to hack pypy you are going to get bogged down in the translation times.

Case 1 - vmprof

I want to guide you through some of the projects. Some are more finished than others, but they share in common some cross-cutting of the layers, which is often the only way to get the absolute best of the performance. vmprof is a high performance statistical profiler that we once thought could be spun into a business. Vmprof is a set of hints that you can apply to any interpreter - using simple decorators. Then it's used in the pypy interpreter.

Case 2 - cffi backend

Another example - cffi is a lightweight way to call C from Python. In PyPy our aim is to have a minimum call - just an assembler call if converting arguments is not too complicated. Let's walk through various layers how this is peeled to present a good interface and optimize things in the process

Case 3 - utf8

One of the recent pieces of work I've been doing is using internally utf8 for storing unicode as opposed to UCS4. CPython has a hybrid approach of storing either ascii, ucs2 or ucs4 starting from python 3.3 (or 3.5?). UTF8 is almost always more compact, especially for mixed content, encoding/decoding is essentially free and indexed access can be made fast. More compact = faster, in case of dynamic languages

exercise

Todays exercise - implement cffi.temp_utf8(unicode) on utf8 branch (XXX maybe insert fixed revision) which will provide a context manager that lets you access utf8 buffer without copying for a limited scope, looking directly into GC object. I'll explain the basics on how to get access to interior of something in RPython.

code4