Clément Béra ~ Smalltalk, Tips 'n Tricks

FullBlockClosure design

27 Monday Jun 2016

Posted by Clement Bera in Cog, Pharo

≈ **Leave a comment**

Hi everyone,

As discussed in the previous post I am attempting to introduce FullBlockClosure as a replacement for BlockClosure in Pharo.

The introduction of the FullBlockClosure leads to simplification in Cog's JIT and other part of the system related to bytecodes. This is because with the new model activating a closure or a method are much more similar, hence more code can be reused between both.

Let's look at the FullBlockClosure creation bytecode in the SistaV1 bytecode set (which I have been using for a week in Pharo without crashes):

```
* 255 11111111 xxxxxxxx siyyyyyy push Closure Compiled block literal index
xxxxxxxx (+ Extend A * 256) numCopied yyyyyy receiverOnStack: s = 1
ignoreOuterContext: i = 1
```

At FullBlockClosure creation time, the VM is aware of:

- The index in the literal frame of the compiledMethod of the compiledBlock (a compiledBlock being very similar to a compiledMethod), which represents the code to execute when the closure is activated.
- The number of copied values.
- A bit marks if the receiver in the closure is the outerContext's receiver, or another object pushed on stack. This is used for inlining.
- A bit marks if the closure needs an outerContext or not (this is used for copying block to avoid object allocation).

The FullBlockClosure created is filled with those fields:

- Field 1: outerContext: the outerContext of the closure if the bytecode creation specifies it, else nil.
- Field 2: (previously startpc) compiledCode: an instance of compiledBlock, which represents the bytecode to execute when the closure is activated.
- Field 3: numArgs: number of arguments of the closure. It's cached here for efficient #cull:

implementation.
- Field 4: receiver: the receiver of the closure activation, which is not necessarily the outerContext receiver.

Upon activation, the FullBlockClosure now fetches:

- The receiver from its receiver field instead of the outerContext receiver field.
- The compiledBlock from its compiledCode field instead of the outerContext method offset by its startpc.

The new design have implications for the (compiler/VM) programmer:

- A full closure's outerContext is now used only for non local returns. Any block without a non local return does not need an outerContext, except for advanced debugging features.
- The compiled block (code the closure has to execute) is now completely independent from the method the closure is created. This means for example that the JIT can compile to machine code and install separately a method and its inner block closures. Other similar things are simplified.
- It's possible to inline a method with a closure in another method without decompiling the closure's bytecode (The decompiler still needs to check for non local return though). In the past we need to update the closure bytecode, specifically access to instance variables or literals, which was tedious and required additional deoptimization metadata.

Hopefully someone will implement copying blocks and clean blocks, and we will see if the performance overhead is important or not.

With this changes not only Cog's JIT but also the runtime optimizer is simplified. I'm integrating the FullBlockClosure changes, it's likely that I will propose an open alpha of the Pharo Sista image (you'll need to compile the Sista VM to experiment) before ESUG. The debugging support won't be complete (each context on optimized code *can* be deoptimized, it's just about having the tools like the debugger doing it), but the benchmarks should run on the alpha version and hopefully some people will help me to integrate new optimization passes to reach the expected 3x performance boost.