Virtual CPU

ESUG, Cambridge 2014

By Igor Stasenko & Max Mattone
RMod, Inria

Highlights

- What?
- Why?
- How?
- Demo
- To Do

What is VCpu?

- a framework to write low-level code
- can simulate & generate machine code
- multiple backends for ARM, x86/x64 code generation
- 100% implemented in smalltalk

What is VCpu NOT

- NOT a full-fledged compiler with numerous data “types”, like GCC/ LLVM
- NO direct support of calling convention(s)
- it is a bare-bone model of computer with CPU & memory.. to build on top of it

Requirements

- expressive power of smalltalk
- malleable
- extensible
- simple
- yet powerful

Architecture

Platform neutral

RTL → VCPU Interface

VCPU Interface → Simulating CPU

VCPU Interface → Generating CPU

Platform dependent

Generating CPU → Low-level intermediate

Liveness analysis

Register allocation

Optimizations

Low-level intermediate → AsmJIT

Why

- there's no low-level compilers for smalltalk
- on inventing own wheel: adapting existing solutions costly as (h/w)ell
- lets us learn as we do it

Don’t mode me in

Implementing low-level semantic

Idea ⟶ Smalltalk ⟶ Slang ⟶ C ⟶ Machine

Idea ⟶ Machine

A more correct picture

Idea $\rightleftharpoons$ Machine

At the end of the day

memory at: x put: y

So, tell me, why you have to be expert in 10+ disciplines to do that?

There must be a better way

- i was looking for a nice & simple solution since 2006
- VCpu interface, is product of number of painful attempts to get there
- Now i am pleased (almost).

How

- imperative rather than declarative
- coding with VCpu is just plain smalltalk

Imperative

- you don't 'compile' or translate code, you just execute own code istructing CPU what to do: `cpu doThat`

Dual nature

- can be either simulated or generating machine code, just use different CPU
- you free to choose any style you want to program it

Machine word

- a facade object representing a virtual CPU register/variable (machine word)
- it easy to manipulate with, since one can define a usual arithmetic operations, like #+ , #- , #* , #/ etc..
- serves as a basis of VCpu ‘DSL’

Lets learn a new DSL

Step 1. Creating a new machine word

```
word := cpu word: 10.
```

Step 2. Assigning new value to existing one

```
word1 := cpu word: 10.
```

```
word2 := cpu word: 4.
```

...

```
word2 value: word1.
```

Right

```
word2 := word1.
```

Wrong!!!

Step 3. Arithmetic expressions

```
word := x + y bitAnd: z
```

just keep in mind, it is not ‘school’ but ‘CPU’ math

Step 3. Arithmetic expressions

the result of expression is always new machine word

```
word := x + y bitAnd: z
```

just keep in mind, maybe you wanted:

```
word value: x + y bitAnd: z
```

... instead

Step 3. Arithmetic expressions

.. expressions can be intermixed with regular constants

```
word := x + 5
```

as long as receiver is machine word

Step 4. Memory access

```
word := address loadWord
```

```
address writeWord: x
```

Step 5. Comparisons/ control flow

```
a>b ifTrue: [..] ifFalse: [..]
```

```
a to: b do: [:i |..]
```

```
x timesRepeat: [ .. ]
```

Looks familiar?

Thanks, we have Opal

- disable inlining (ifTrue:/to:do: ..etc)

```
expression ifTrue: [..] ifFalse: [..]
```

~~mustBeABoolean~~

this is ~~Sparta~~ Pharo!

Step 6. Call/return

address call

cpu return.

cpu return: x * *

* * requires a notion of calling convention

End of tutorial

NativeBoost integration

NativeBoost-style callout

```
abs: x
    <primitive: #primitiveNativeCall module: #NativeBoostPlugin>
    ^ self nbCallout
        function: #( int abs (int x ) )
        module: NativeBoost CLibrary
```

VCpu-style callout

?

NativeBoost integration

NativeBoost-style callout

```
abs: x
    <primitive: #primitiveNativeCall module: #NativeBoostPlugin>
    ^ self nbCallout
        function: #( int abs (int x ) )
        module: NativeBoost CLibrary
```

VCpu-style callout

```
abs: x
    <primitive: #primitiveNativeCall module: #NativeBoostPlugin>
    ^ self nbCallout
        function: #( int abs (int x ) )
        module: NativeBoost CLibrary
```

this is ~~Sparta~~ Pharo!

Demo

To Do

- Finish optimizations
- Test ARM Support (on real hardware)
- Complete NativeBoost VCpu implementation
- Documentation
- Look forward for Spur integration

Wednesday, August 20, 14