2017 FAST Conference
La Plata, Argentina
November, 2017

VA Smalltalk 9: Exploring the next-gen LLVM-based virtual machine

Alexander Mitin
Senior Software Engineer
Instantiations, Inc.

instantiations
VA Smalltalk

The New VM Runtime

- 64-bit & 32-bit
- Written from scratch
- Compatibility & Performance
- LLVM is a core tech

instantiations
VA Smalltalk

Traditional Way

- Written in C or C++
  - Interpreter is a huge case statement
  - Or a huge function with lots of labels
- Lots of overhead
- Support for JIT is complicated

Options?

- Continue to develop using legacy tools?
- Use assembly language?
- Develop a special language for building a VM?

Let's C

A typical C language compiler has these stages:

- Preprocessor
- Code parsing
- Building some intermediate representation
- Optimizations
- Machine code generation
- Output to an object file

Let's C

A typical C language compiler has these stages:

- ~~Preprocessor~~
- ~~Code parsing~~
- ~~Building some intermediate representation~~
- Optimizations
- Machine code generation
- Output to an object file

instantiations
VA Smalltalk

LLVM Can Do it

LLVM is a compiler infrastructure

- Advanced IR
- Object file generation
- Open-source with big community
- Capable for JIT

Instantiations Way

- Written in LLVM IR
  - Every bytecode handler is a function
  - Fine-tuned using LLVM intrinsics and attrs
- Minimized overhead
- Transparent interpreter-JIT transitions

instantiations
VA Smalltalk

A bytecode

```
01 I_32 BCpushMagicB(VMContext* vmStruct, oop** sp, I_8* pc)
02 {
03     I_PTR operand = (I_PTR)*pc++;                                    // load a bytecode
operand
04     *--sp = (oop*)operand;                                           // store it to a
stack slot and advance sp
05     I_8 nextBC = *pc++;                                                      // get
next bytecode
06     BCFuncType* nextHandler = bytecodeTable[nextBC];// get next bytecode handler function
07     return nextHandler(vmStruct, sp, pc)                        // call it
08 }
```


instantiations
VA Smalltalk

LLVM Crash Course

- Modules & Functions
- Basic Blocks
- SSA form

instantiations
VA Smalltalk

A bytecode

```
01 define internal cc18 i32 @BCpushMagicB(%VMContext* inreg %vmStruct, %oop** inreg %sp, i8* inreg %pc) #1 align 8
02 {
03  %1 = load i8, i8* %pc, align 1
04  %2 = sext i8 %1 to i64
05  %3 = getelementptr inbounds i8, i8* %pc, i64 1
06  %4 = inttoptr i64 %2 to %oop*
07  %5 = getelementptr %oop*, %oop** %sp, i64 -1
08  store %oop* %4, %oop** %5
09  %6 = load i8, i8* %3
10  %7 = zext i8 %6 to i64
11  %8 = getelementptr inbounds i8, i8* %3, i64 1
12  %9 = getelementptr inbounds @bytecodeTable, i64 0, i64 %7
13  %10 = load BCFuncType*, BCFuncType** %9, align 8
14  %11 = tail call cc18 i32 %10(%VMContext* inreg %vmStruct, %oop** inreg %5, i8* inreg %8)
15  ret i32 %11
16 }
```


instantiations
VA Smalltalk

A bytecode

```
01 void EsVMBytecodes::pushMagic(IRBuilder<>& b, Value* sp, Value* pc, Value* bp, EsBCOpType
bcot)
02 {
03         Value *operand;
04         Value *currentPC = getNext(b, pc, bcot, operand);
05         Value *currentSP = stackPush(b, sp, operand);
06         executeNextBytecode(b, currentSP, currentPC, bp);
07 }
```


instantiations
VA Smalltalk

A bytecode

```
00 BCpushMagicB:
01          movsbq      (%rsi), %rax                    ; load bytecode operand from pc
02          movq                    %rax, -8(%r15)                  ; store it into a stack slot
03          addq                    $-8, %r15                       ; advance stack pointer
04          movzbl      1(%rsi), %eax                   ; get next bytecode from pc + 1
05          addq                    $2, %rsi                        ; advance pc by 2
06          movabsq     $bytecodeTable, %rcx
07          rex64 jmpq *(%rcx,%rax,8)                   ; indirect jump to the next bytecode
handler
```


instantiations
VA Smalltalk

The Idea

- LLVM tail-call elimination optimization
- Keep performance-critical data in CPU regs

Method Exec: Interpreted

```
// a pseudo-code
I_32 methodExec(VMContext*, I_PTR arg1, I_PTR arg2)
{
    AllocateStackFrame(ftInterpreted);
    return ExecuteNextBytecode(arg2); // tail-call
}
```


instantiations
VA Smalltalk

Method Exec: Native

```
// a pseudo-code
I_32 methodExec(VMContext*, I_PTR arg1, I_PTR arg2)
{
    AllocateStackFrame(ftNative);
    // execute JIT-generated native code in place
}
```


instantiations
VA Smalltalk

Memory Manager

- Written in C from scratch
- So far the same algorithm as it was in v.8
- It’s ready for modifications

Build System

- CMake based
- MinGW64/gcc compiler
- A special app with LLVM IR
- MinGW64/gcc linker

instantiations
VA Smalltalk

Future Work

- Unicode support
- Alternative memory manager with parallel GC
- Alternative JIT engine (OMR?)
- Make use of CPU’s stack pointer register
- Compressed OOPs

Thank you for your attention

Questions?

Contacts:

E-mail: amitin@instantiations.com

GitHub: amitin

instantiations
VA Smalltalk