Low Level Virtual Machine and BrainF

Sterling Stuart Stein
stein @ ir.iit.edu

September 30, 2007

LLVM

- Can be used to make compilers and virtual machines
- Can generate code:
  - Interpreted
  - Native
  - Statically (ordinary executable)
- Is a compiler backend
- Supports many different platforms

Compiler structure

- Can have multiple frontends:
  C, C++, ObjC, Fortran, Java, etc.
- Can have multiple backends:
  x86, AMD64, PPC, ARM, MIPS, C, etc.
- LLVM uses intermediate representation

$$\begin{tikzcd}
C \arrow[dr] & & x86 \\
C++ \arrow[r] & IR \arrow[ur] \arrow[r] \arrow[dr] & AMD64 \\
Fortran \arrow[ur] & & PPC
\end{tikzcd}$$

Intermediate representation

- Like assembly, but still target independent
- Doesn't know about calling conventions, size of pointers, etc.
- Typed: i1, i8, i32, i8 *, float, structs, function pointers, ...

Basic blocks

- You enter at the top and leave at the bottom
- Ends with a branching instruction (br, ret)

```
// C
if (x) {
  f();
} else {
  g();
}
return 0;
```

$\rightarrow$

```
; LLVM assembly
block.0:
  br i1 %x, label %block.1, label %block.2

block.1:
  call void @f()
  br label %block.3

block.2:
  call void @g()
  br label %block.3

block.3:
  ret i32 0
```

Static Single Assignment

- Every variable is assigned exactly once
    - Use subscripts on name for versions of 1 variable
    - Think of as identifying statements instead of variables
- For multiple in-edges, use phi nodes

```
%head.4 = phi i8*
    [%head.3,%main.2],[%head.5,%main.4]
```

- Equivalent to continuation passing style (CPS)
- Makes optimizations simpler

BrainF

- Implemented BrainF compiler in LLVM
- Based on Turing machine, has head and tape
- Minimalistic language: 8 commands:

BrainF	C	Action
,	*h=getchar();	Read a character from stdin
.	putchar(*h);	Write a character to stdout
-	- -*h;	Decrement tape
+	++*h;	Increment tape
<	- -h;	Move head left
>	++h;	Move head right
[	while(*h) {	Start loop
]	}	End loop

BrainF snippets

- Plain text
  Comments
- [Whole sentence comment, including commas and periods.]
  More complex comments
- [−]
  Set cell to 0
- + + + + + + + + + + [> . < −]
  for(i=0;i<10;i++) {putchar(x);}
- + + + + + + + + [< − − − − −− > −]
  Subtract 48
- −[> . < [−]]
  if (x!=1) {putchar(y);}

BrainF in LLVM

- Translate commands into LLVM

BrainF	LLVM

Header	declare void @llvm.memset.i32(i8*, i8, i32, i32)
	declare i32 @getchar()
	declare i32 @putchar(i32)
	
	define void @brainf() {
	brainf.0:
	  %arr = malloc i8, i32 65536
	  call void @llvm.memset.i32(
	    i8* %arr, i8 0, i32 65536, i32 1)
	  %head.0 = getelementptr i8* %arr, i32 32768

Footer	brainf.1:
	  free i8* %arr
	  ret void
	}

BrainF in LLVM

- Translate commands into LLVM

BrainF	LLVM
+	%tape.0 = load i8* %head.0 %tape.1 = add i8 %tape.0, 1 store i8 %tape.1, i8* %head.0
-	%tape.0 = load i8* %head.0 %tape.1 = add i8 %tape.0, -1 store i8 %tape.1, i8* %head.0
<	%head.1 = getelementptr i8* %head.0, i32 -1
>	%head.1 = getelementptr i8* %head.0, i32 1
.	%tape.0 = load i8* %head.0 %tape.1 = sext i8 %tape.0 to i32 call i32 @putchar(i32 %tape.1)
,	%tape.0 = call i32 @getchar( ) %tape.1 = trunc i32 %tape.0 to i8 store i8 %tape.1, i8* %head.0

BrainF in LLVM

- Translate commands into LLVM

BrainF	LLVM

[	
	```
	    br label %looptest.0

	loopbody.0:
	```

]	
	```
	    br label %looptest.0

	looptest.0:
	  %head.2 = phi i8* [ %head.0, %loopbefore.0 ],
	    [ %head.1, %loopbody.0 ]
	  %tape.0 = load i8* %head.2
	  %test.0 = icmp eq i8 %tape.0, 0
	  br i1 %test.0, label %loopafter.0, label %loopbody.0

	loopafter.0:
	```

Conclusion

- Have working BrainF compiler and interpreter
- Runs on multiple platforms
- Runs very quickly because of LLVM's many optimizations
  - Can convert loads and stores into registers
- LLVM is easy to work with and is a powerful tool