A Comparison of API-based JIT Compiler Overhead during Run-time

Eric Coffin
Faculty of Computer Science
University of New Brunswick
eric.coffin@unb.ca

Background

- Just-in-Time (JIT) compilation can provide significant performance gains for applications.
- Rather than writing a JIT compiler from scratch why not embed an existing one?
- Two popular embeddable JIT frameworks are LLVM MCJIT and Eclipse OMR JitBuilder.

LLVM - MCJIT

- Designed for life-long application analysis and transformation.
- IR is in SSA form organized in basic blocks
- 31 RISC style opcodes
- JIT compiler is MCJIT which shares code with front-end.
- Can perform 40 analysis and 60 transform passes.

```
define i32 @mul_add(i32 %x, i32 %y, i32 %z) {
    entry:
    %tmp = mul i32 %x, %y
    %tmp2 = add i32 %tmp, %z
    ret i32 %tmp2
}
```

LLVM IR for function returning x * y + z

JitBuilder

- Embeddable JIT API for Eclipse OMR
- Eclipse OMR provides many runtime components for OpenJ9
- IR is a DAG of nodes (called trees)
- Unorderable nodes belong to a list called tree-tops
- Nodes can be reused to simplify optimizations such as CSE
- TRJIT offers 170 optimizations (limited in JitBuilder)

```
treetop--> istore a
           |
           imul--- isub-------+
           |                  |
           iadd               |
           |                  |
           +-----> iload b <--+
           |                  |
           +-----> iload a <--+
}
```

OMR IR for (a+b)*(a-b)

Test Programs

- Embedded frameworks with test programs
  - Increment: add 1 to an integer argument
  - Recursive-fib: recursive fibonacci
  - Iterative-fib: iterative Fibonacci
- Measured memory overhead, CPU time, disk space, and usability.
- Ubuntu 19.04, 32GB RAM, Intel i7-8700 CPU (6 core)

Results

	LLVM MCJIT			Eclipse OMR JitBuilder		
Program	mean (ns)	median	std. dev.	mean (ns)	median	std. dev.
increment	1,378,060	1,387,338	39,774	536,741	537,103	2620
recursive-fib	1,548,316	1,547,491	3925	1,854,393	1,850,322	21,754
iterative-fib	2,509,502	2,519,808	60,429	4,192,213	4,191,730	10,419

Results of compiling each function 20 times with each framework.

Program	LLVM MCJIT	Eclipse OMR JitBuilder
increment	85,250	0
recursive-fib	12,126,516	26,852,988
iterative-fib	148,026	35,754

Estimated time to execute JIT-ed function 1000 times.

Program	LLVM MCJIT	Eclipse OMR JitBuilder	Native (C++)
increment	60,640,216	10,148,608	16,616
recursive-fib	60,671,176	10,149,272	16,600
iterative-fib	60,652,728	10,149,184	16,600

Total size in bytes of linked binary test programs.

	LLVM MCJIT	Eclipse OMR JitBuilder	Native (C++)
Program	max RSS (kb)	max RSS (kb)	max RSS (kb)
increment	43,544	9416	1548
recursive-fib	45,472	9768	1532
iterative-fib	44,452	10,044	1684

Max RSS during a single compilation and execution.

Summary

- LLVM generated JIT function faster in most cases
- LLVM generated code executed faster in most cases
- JitBuilder memory and disk overhead significantly smaller
- LLVM was challenging to integrate
  - IR was more difficult to program
  - Linking was a major challenge (160 objects in LLVM)
- JitBuilder was simple to integrate and program IR
- JitBuilder needs more configuration (locked at warm level)