Y JIT: a basic block versioning JIT compiler for CRuby

Akira Kawata

1 Introduction

2 Background

2.1 The Ruby Programming Language

- Rubyの最適化はむつかしい
  - Rubyの関数は動的に再定義が可能
    - 整数同士のpritimitiveな演算も含む
  - 第一級環境がサポートされている
    - クロージャのことか?
    - ローカル変数がスコープの外から変更できる

2.2 The CRuby Virtual Machine

- CRubyのVMはYARV
  - 笹田耕一 さんが開発
  - スタックマシン
    - 2本のスタックがある
    - 命令セット
- 小さな即値についてはタグビットをつけて直接埋め込む
- 関数呼び出しを多用し、11種類もある

type	Index	Instruction	Operands	Stacks		
nop	0	nop			->	
variable	1	getlocal	idx		->	val
	2	setlocal	idx	val	->	
	3	getspecial	idx, type		->	val
	4	setspecial	idx	obj	->	
	5	getdynamic	idx, level		->	val
	6	setdynamic	idx, level	val	->	
	7	getinstancevariable	id		->	val
	8	setinstancevariable	id	val	->	
	9	getclassvariable	id		->	val
	10	setclassvariable	id, decip	val	->	
	11	getconstant	id	klass	->	val
	12	setconstant	id	val, klass	->	
	13	getglobal	entry		->	val
	14	setglobal	entry	val	->	
put	15	putnil			->	val
	16	putself			->	val
	17	putundef			->	val
	18	putobject	val		->	val
	19	putstring	val		->	val
	20	concatstrings	num	...	->	val
	21	tostring		val	->	val
	22	toregexp	flag	str	->	val
	23	newarra	num	...	->	val
	24	duparra	ary		->	val
	25	expandarray	num, flag	..., ary	->	...
	26	concatarray		ary1, ary2st	->	ary
	27	splatarra	flag	ary	->	obj
	28	checkincludearray	flag	obj, ary	->	obj, result
	29	newhash	num	...	->	val
	30	newrange	flag	low, high	->	val
	31	putnot		obj	->	val
stack	2	pop		val	->	
	3	dup		val	->	val1, val2
	4	dupn	n	...	->	...
	5	swap		val, obj	->	obj, val
	6	reput		..., val	->	val
	7	topn	n	...	->	val
	38	emptstack		...	->	...
setting	39	definemethod	id, body, is_singleton	obj	->	
	40	alias	v_p, id1, id2		->	
	41	undef	id		->	
	42	defined	type, obj, needstr	v	->	val

2.3 MJIT

- CRubyの最初のJITコンパイラ
- 実行中にCの関数を生成しコンパイルし読み込む
- 実行中にCコンパイラが呼ばれる
- 直接機械語を生成するわけではないので細かい最適化ができない
- 現在masterからは消えている気がする
  - mjit.cがない
- 最近RJITというのも追加された

2.4 Lazy Basic Block Versioning

- YJITが使っているコンパイル手法
- ベーシックブロックごとにコンパイルする
  - メソッドベースのコンパイラはメソッドを丸ごとコンパイルする
  - 分岐がある場合は通った分岐だけコンパイルする
  - 通らなかった分岐にはスタブをおいておく
- 実行されないコードはコンパイルしない

3 Architecture of YJIT

3 Architecture of YJIT

- x86-64上のLinuxとMac OSをサポート
- ShopifyとGitHubのワークロードを動かすため

Maxime Chevalier-Boisvert
Shopify
Canada
maxime.chevalierboisvert@shopify.com

Noah Gibbs
Shopify
United Kingdom
noah.gibbs@shopify.com

Jean Boussier
Shopify
France
jean.boussier@shopify.com

Si Xing (Alan) Wu
Shopify
Canada
paper@alanwu.email

Aaron Patterson
Shopify
United States
aaron.patterson@shopify.com

Kevin Newton
Shopify
United States
kevin.newton@shopify.com

John Hawthorn
GitHub
Canada
john@hawthorn.email

3.1 Integration with CRuby (1)

- ある関数がCRubyのdirect-threaded interpreter(=YARV VMのこと?)から10回呼ばれるとYJITのコード生成が始まる
  - yjit/src/codegen.rs#L758-L926
- 分岐が含まれていた場合はスタブを含んだ機械語が生成される

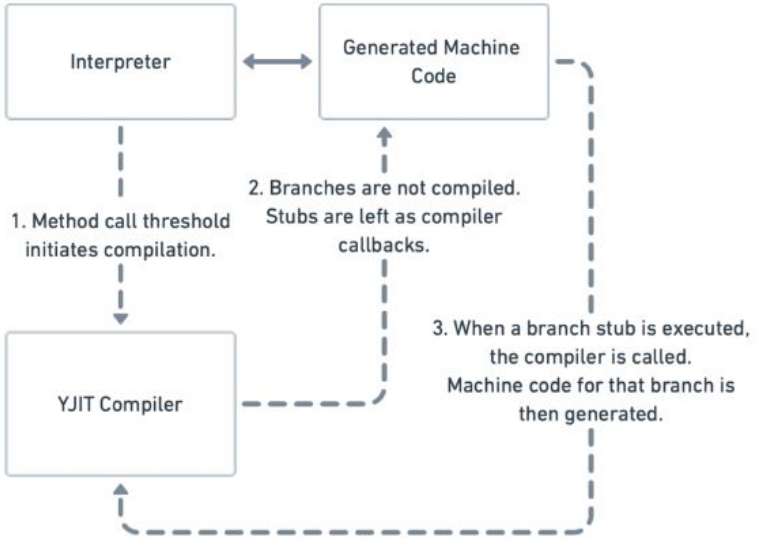

Figure 1. YJIT Compilation Pipeline.

3.1 Integration with CRuby (2)

- YJITはCRubyの全てのバイトコードサポートしていない
  - サポートしていない場合はインタプリタに戻る
  - yjit/src/codegen.rs#L871-L890
- 簡単のためYJITはCRubyと同じ呼び出し規約とスタックフレームレイアウトを使う
  - レジスタは使わない
  - CRubyのGCと相性が良い

3.2 Deoptimization

- YJITは生成したコードを細かい単位で無効化できる
  - コード生成時の前提が無効になった場合はこの単位で無効化する
    - 整数演算が再定義されたり

3.3 Run-Time Value Promotion

- 実行時の値をコンパイル時定数として扱う機能
  - 実行時の値が決まったらコンパイルを再開することで実現
  - Psyco VM、PyPyでも使われている機能
    - Psyco is an unmaintained specializing just-in-time compiler for pre-2.7 Python
- 多層なインラインキャッシュを実現するのに使われている
  - 型ごとに機械語を出しわけるということか?

3.4 Type Specialization

- 型情報を追跡し最適化に使っているが効果はわかっていない
- CRubyがボトルネックになっている
  - 関数呼び出し
  - 変数アクセス

Table 1. Basic types tracked by YJIT.

Type	Description
IMM	Immediate value of unknown type
FALSE	The immediate value `false`
NIL	The immediate value `nil`
TRUE	The immediate value `true`
FIXNUM	Immediate 63-bit signed integer
FLONUM	Immediate floating point value
STATIC_SYMBOL	Immediate Ruby symbol
HEAP	Heap-allocated object of unknown type
ARRAY	Dynamically-extensible list
HASH	Map of keys to values
STRING	Character string

4 Evaluation

4.1 Performance on Benchmarks

- TruffleRubyにrailsbenchに負けている
- YJITがTruffleRubyより良い理由がいろいろ書いてある

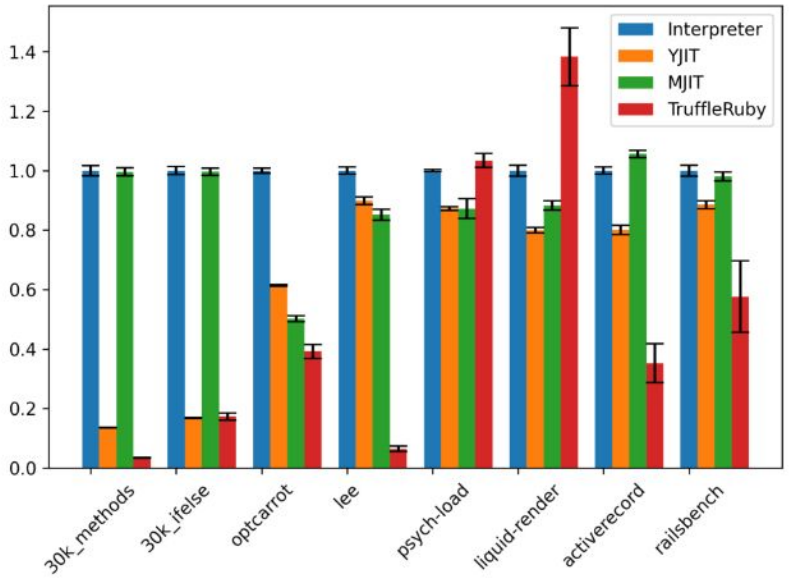

Figure 2. Mean execution time on benchmarks, normalized to the time taken by the interpreter (lower is better).

5 Related Work

5 Related Work

- TruffleRuby
  - Truffle/Graal 上に構築している
  - JVMってそんなに速いんですか?
- Stripe’s Sorbet Compiler
  - ストライプはオンライン決済の会社
  - Ahead of Timeコンパイラを作っている
  - Ruby on Railsの会社はみんなRubyの効率化に手を出す

6 Future Work

6 Future Work

- Faster Calling Convention
  - 関数呼び出しのコストが支配的らしい
- Object Shapes
  - 変数アクセスも遅い
  - モダンな動的言語はObject Shapesを使ってこれを解決している

(どちらもYJITではなくてCRubyの欠点)

Source code reading

yjit ディレクトリより

- yjit/src/codegen.rs#L8164-L8269
  - Maps a YARV opcode to a code generation function (if supported)
- yjit/src/core.rs#L1106-L1256
  - GC callbacks
- yjit/src/disasm.rs
- yjit/src/virtualmem.rs
  - アドレスを事前に確保しておいてmprotectで実際に使うところを有効にしていく

参考

- https://github.com/ruby/ruby/blob/master/doc/yjit/yjit.md
- https://dl.acm.org/doi/10.1145/3486606.3486781