# LLVM Backend for HHVM

**Brett Simmers**
**Maksim Panchenko**

Facebook

# HHVM

## JIT for PHP/Hack

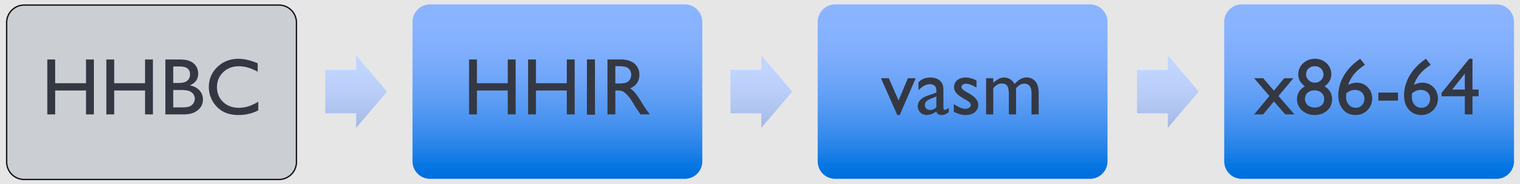
- Initial work started in early 2010
- Running facebook.com since February 2013
- Open source! http://hhvm.com/repo
- wikipedia.org since December 2014
- Baidu, Etsy, Box, many others: https://github.com/facebook/hhvm/wiki/Users

# HHVM

## JIT for PHP/Hack

- Not a PHP -> C++ source transformer: that was HPHPc.
- Emits type-specialized code after verifying assumptions with type guards.
- Ahead-of-time static analysis eliminates many type guards, speeds up other operations as well.
- 2-4x faster than PHP 5.6: http://hhvm.com/blog/9293/lockdown-results-and-hhvm-performance

# HHVM Compilation Pipeline




# Modifications to HHVM

## PHP Function Calls

- No spilling across calls – native stack is shared between all active PHP frames.
- Callee may leave jitted code, interpret for a while, and resume after bindcall instruction.
- No support for catching exceptions – pessimizes many optimizations.

- Fixed all limitations and implemented using invoke instruction – also helped existing backend.

# Modifications to HHVM

## Generalizing x86-specific concepts in vasm

- idiv: %rax and %rdx are implicit inputs/outputs.
- x86-64 implicitly zeros top 32 bits of registers.
- Endianness: had to shake out any assumptions of a little-endian target.

# Codegen Differences

## Arithmetic Simplification

### vasm

```
movq  -0x20(%rbp), %rax
mov %rax, %rcx
shl $0x1, %rcx
... 11 more lines of shl/add ...
add %rdx, %rcx
mov %rax, %rdx
shl $0x28, %rdx
add %rdx, %rcx
add %rcx, %rax
movb  $0xa, -0x18(%rbp)
movq  %rax, -0x20(%rbp)
```

### LLVM

```
mov $0x100000001b3, %rax
imulq  -0x20(%rbp), %rax
movb  $0xa, -0x18(%rbp)
movq  %rax, -0x20(%rbp)
```

# Codegen Differences

## Tail Duplication

### vasm

```
0x0: callq ...
0x1: test %rax, %rax
0x2: jnz 0x5
0x3: mov $0x0, %al
0x4: jmp 0x9
0x5: cmpb  $0x50, 0x8(%rax)
0x6: cmovzq  (%rax), %rax
0x7: cmpb  $0x8, 0x8(%rax)
0x8: setnle %al
0x9: test %al, %al
0xa: jz ...
0xb: jmp ...
```

### LLVM

```
0x0: callq ...
0x1: test %rax, %rax
0x2: jz ...
0x3: cmpb  $0x50, 0x8(%rax)
0x4: cmovzq  (%rax), %rax
0x5: cmpb  $0x9, 0x8(%rax)
0x6: jl ...
0x7: jmp ...
```

# Codegen Differences

## Misc

- Large switch statements: single path of comparisons vs. binary search.
- Register allocator: sometimes vasm spills fewer values, sometimes LLVM. LLVM generally better at avoid reg-reg moves.
- vasm almost always prefers smaller code due to icache pressure. Bad for microbenchmarks, good for our workload.

# LLVM Changes

## Correctness and Performance

- Custom calling conventions
- Location records
- Smashable call attribute
- Code size optimizations
- Performance tweaks

# Calling Conventions

## Correctness

- VMs SP and FP pinned to %rbx and %rbp
- %r12 used for thread-local storage
- Different stack alignment for *hhvmcc*
- C++ helpers always expect VmFP in %rbp
- 5 calling conventions + more planned

# (Almost) Universal Calling Convention

- Can use any number of regs for passing arguments
- Pass *undef* in unused regs
- Can return in any of 14 GP registers
- %r12 still reserved and callee-saved
- 5 -> 2 calling conventions

# Location Records

## Correctness

- Replace destination of *call/jmp* after code gen
- Locate code for a given IR instruction (*call/invoke*)
- Why not use *patchpoint?*
- Support tail call optimization
- Use direct call instruction
- Don’t need de-optimization information

# Location Records

## Correctness

- `musttail call void @foo(i64 %val), !locrec !{i32 42}`
- Propagate info to MCInst
- Data written to .llvm_locrecs
- Unique ID per module
- Works with any IR instruction
- Switch from metadata to operand bundles

# Call with LocRec

## Example

```
$ cat smashable.ll
   ...
   %tmp = call i64 @callee(i64 %a, i64 %b) !locrec !{i32 42}
   ...
$ llc < smashable.ll
   ...
   .Ltmp0:                                 # !locrec 42
        pushq   %rax
   .Ltmp1:                                 # !locrec 42
        callq   callee
```

# Call with LocRec

## Section Format

```
   .section .llvm_locrecs
...
        .quad   .Ltmp0  # Address
        .long   42      # ID
        .byte   1       # Size
        .byte   0
        .short  0
        .quad   .Ltmp1  # Address
        .long   42      # ID
        .byte   5       # Size
        .byte   0
        .short  0
```

# Smashable Call Attribute

## Correctness Change

- Overwrite destination in MT environment after code generation and during code execution
- Instruction shall not pass 64-byte boundary
- Use modified .bundle_align_mode
- Works with *call/invoke* only

# Smashable Call with LocRec

## Example

```
$ cat smashable.ll
   ...
   %tmp = call i64 @callee(i64 %a, i64 %b) smashable, !locrec !{i32 42}
   ...
$ llc < smashable.ll
   ...
   .Ltmp0:                                 # !locrec 42
        pushq   %rax
        .bundle_align_mode 6
   .Ltmp1:                                 # !locrec 42
        callq   callee
        .bundle_align_mode 0
```

# Code Skew

## Correctness Change

- Smashable needs 64-byte boundary
- JIT does not know where the code goes
- JIT has to request 64-byte aligned code section?
- Our code is packed
- Use “code_skew” module flag to modify effect of align directives

# HHVM+LLVM Checkpoint

## Correctness Done

- 80% coverage
- -10% performance
- Increase coverage
- Increase performance

# Size & Performance Tweaks

## Performance

- Eliminate relocation stubs
- Allow no alignment for any function
- Code gen tweaks for size
- No silver bullet
- “-Os” vs “-O2” not much difference

# Code Splitting

## Performance

- Profile- and heuristic-driven basic block splitting
- 3 code blocks: hot/cold/frozen
- Improved I$ and iTLB performance
- Hacky implementation was easy
- C++ exception support required runtime mods

# Tail call via *push+ret*

## Performance

- Enter PHP function via call
- No return address on stack - use tail call to return
- Makes HW return buffer unhappy
- Could not use *patchpoint* since has to be after epilog
- Custom call attribute *TCR* to force *push+ret*
- Net worth: ~1.5% CPU time

# Code Size

```
; Common pattern — decrement ref counter and check

%t0 = load i64, i64* inttoptr (i64 60042 to i64*)
%t1 = sub nsw i64 %t0, 1
store i64 %t1, i64* inttoptr (i64 60042 to i64*)
%t2 = icmp sle i64 %t1, 0
br i1 %t2, label %l1, label %l2
```

# llc < decmin.ll

```
movq    60042, %rax
leaq    -1(%rax), %rcx
movq    %rcx, 60042
cmpq    $2, %rax
jl      .LBB0_2
```

# Code Size

```
; Common pattern — decrement counter

%t0 = load i64, i64* inttoptr (i64 60042 to i64*)
%t1 = add nsw i64 %t0, -1
store i64 %t1, i64* inttoptr (i64 60042 to i64*)
%t2 = icmp sle i64 %t1, 0
br i1 %t2, label %l1, label %l2
```

# llc < decmin.ll

```
decq    60042
jle     .LBB0_2
```

# llc < decmin.ll

```
decq    60042
jle     .LBB0_2
```

# opt -O2 -S | llc

```
movq    60042, %rax
leaq    -1(%rax), %rcx
movq    %rcx, 60042
cmpq    $2, %rax
jl      .LBB0_2
```

# Conditional Tail Call Optimization

```
func() {
  if (cond)
    return foo();
  else
    return bar();
==================================
  cmpl  %esi, %edi
  jg  .L5
  jmp bar
.L5:
  jmp foo
```

# Conditional Tail Call

```
func() {
  if (cond)
    return foo();
  else
    return bar();
=================================
  cmpl  %esi, %edi
  jg  foo
  jmp bar

; How much win!?
```

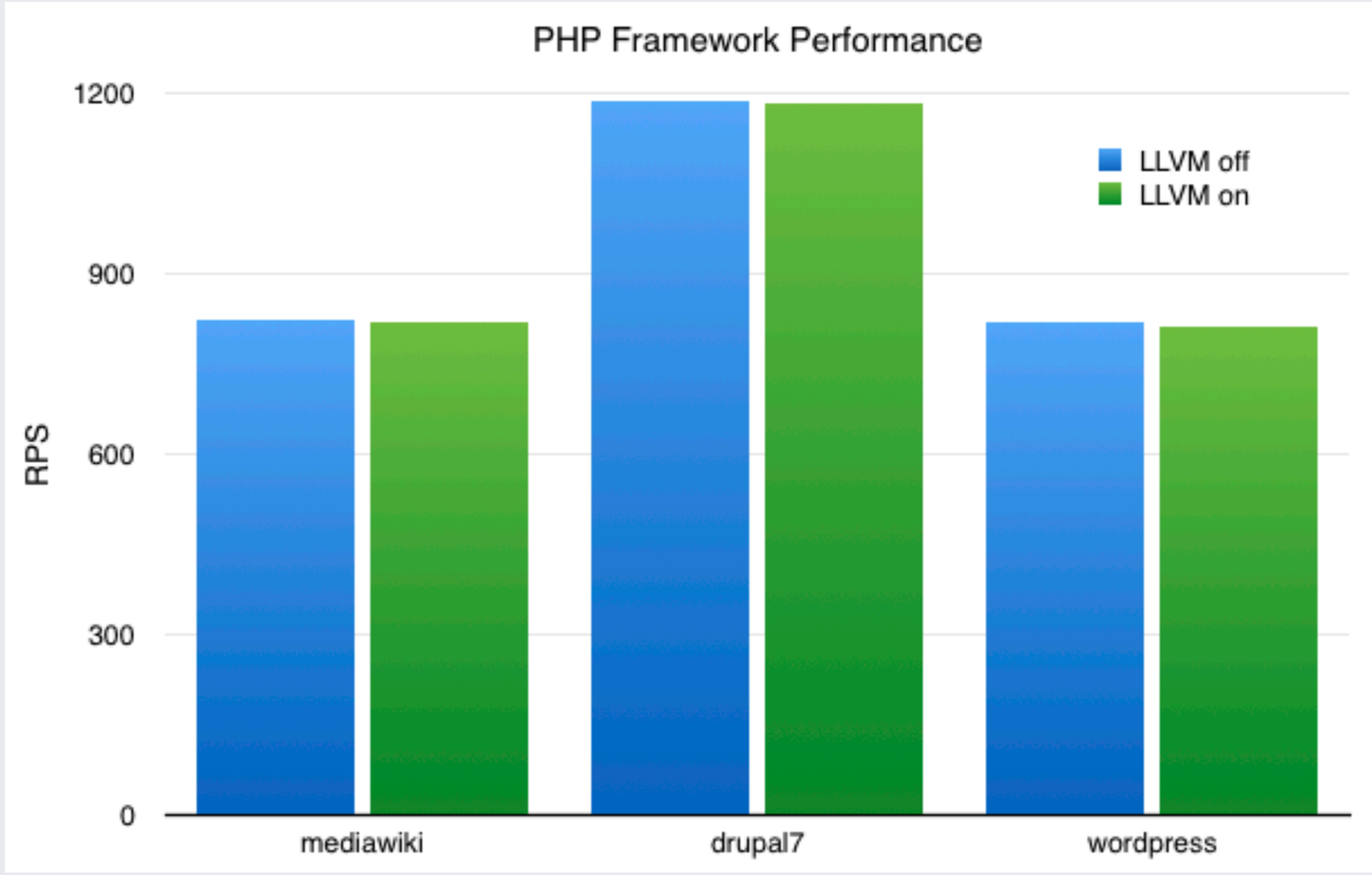
# Conditional Tail Call

```
; BAD order ~50% slowdown
  foo:
  bar:
  func:

; GOOD order ~30% win
  func:
  foo:
  bar:
```

# Performance

## Open Source PHP Frameworks



# Performance

## Facebook Workload

- vasm and LLVM backends not measurably different.
- LLVM clearly beats vasm in certain situations – not hot enough to make a difference overall.
- Not currently using in production – need a reward to take risk.

# Upstreaming Plans

- Patches to LLVM 3.5 are on github (HHVM)
- Calling conventions in LLVM trunk
- Get all required features before 3.8 release
- Switch HHVM to 3.8/trunk LLVM under option

# More Information

http://hhvm.com/
http://hhvm.com/blog/10205/llvm-code-generation-in-hhvm
https://github.com/facebook/hhvm

Freenode: #hhvm and #hhvm-dev