A Practical Solution for Scripting Language Compilers

Paul Biggar, Edsko de Vries and David Gregg

Department of Computer Science and Statistics
Trinity College Dublin

SAC '09: 11th March, 2009
Problem: Scripting languages present “unique” problems (in practice)

Solution: Re-use as much of the *Canonical Implementation* as possible.
A Practical Solution for Scripting Language Compilers

Introduction to phc

Challenges to compilation

phc solution: use the C API

Speedup
1. BSD licence useful since its easy to extend

- Ahead-of-time compiler for PHP
- http://phpcompiler.org
- BSD license
The PHP group claim that they have the final say in the specification of PHP. This group’s specification is an implementation, and there is no prose specification or agreed validation suite. There are alternate implementations [...] that claim to be compatible (they don’t say what this means) with some version of PHP.

A Practical Solution for Scripting Language Compilers

---

Challenges to compilation

---

Batteries included

---

1. all written in C, not PHP
2. Mike Furr earlier: 1000 methods/classes in C
3. 4870 functions, 1000 methods
Change between releases

```php
<?php
    var_dump (0x9fa0ff0b);
?>
```

**PHP 5.2.1 (32-bit)**
```
int(2147483647)
```  

**PHP 5.2.3 (32-bit)**
```
float(2678128395)
```
A Practical Solution for Scripting Language Compilers

Challenges to compilation

Run-time code generation

1. scripting langs are typically made for interpreters
2. can do source inclusion at compile time
3. same mechanism for plugins

<?php
eval ($argv[1]);
?>

<?php
include "mylib.php";
...
include "plugin.php";
...
?>
A Practical Solution for Scripting Language Compilers

Outline

1. Introduction to phc
2. Challenges to compilation
3. phc solution: use the C API
4. Speedup

Introduction to phc
Challenges to compilation
phc solution: use the C API
Speedup
1. RTCG
2. Functions
3. Changes between releases: also use C API at compile-time
1. C API is just zval + macros and functions
2. Use (target) PHP’s C API at run-time

A Practical Solution for Scripting Language Compilers

- phc solution: use the C API

Applicability

- Everything
- Perl
- PHP
- Ruby
- Tcl – I think
A Practical Solution for Scripting Language Compilers

phc solution: use the C API

Applicability

- Everything
  - Perl
  - PHP
  - Ruby
  - Tcl – *I think*

- Except specification
  - Lua
  - Python
A Practical Solution for Scripting Language Compilers

- phc solution: use the C API
- Applicability

Applicability

- Everything
  - Perl
  - PHP
  - Ruby
  - Tcl – I think

- Except specification
  - Lua
  - Python

- Not at all
  - Javascript
A Practical Solution for Scripting Language Compilers

phc solution: use the C API

Simple listings:

```c
// $i = 0;
{
    zval* p_i;
    php_hash_find (LOCAL_ST, "i", 5863374, p_i);
    php_destruct (p_i);
    php_allocate (p_i);
    ZVAL_LONG (*p_i, 0);
}
```
Example: $i = 0

// $i = 0
{
    if (local_i == NULL)
    {
        local_i = EG (uninitialized_zval_ptr);
        local_i->refcount++;  
    }
    zval *p_lhs = &local_i;
    zval **value;
    if (((p_lhs)->is_ref)
    {
        // Always overwrite the current value
        value = *p_lhs;
        zval_dtor (value);
    }
    else
    {
        ALLOC_INIT_ZVAL (value);
        zval_ptr_dtor (p_lhs);
        *p_lhs = value;
    }
    ZVAL_LONG (value, 0);
}
Example: $i = $j

```c
// $i = $j;
if (local_i == NULL) {
    local_i = EG(uninitialized_zval_ptr);
    local_i->refcount++;
}
zval **p_lhs = &local_i;
zval *rhs;
if (local_j == NULL) {
    rhs = EG(uninitialized_zval_ptr);
} else {
    rhs = local_j;
}
if (*p lhs != rhs) {
    if ((*p lhs)->is_ref) {
        // First, call the destructor to remove any data structures
        // associated with lhs that will now be overwritten
        zval_dtor (*p_lhs);
        // Overwrite LHS
        (*p_lhs)->value = rhs->value;
        (*p_lhs)->type = rhs->type;
        zval_copy_ctor (*p_lhs);
    } else {
        zval_ptr_dtor (p_lhs);
        if (rhs->is_ref) {
            // Take a copy of RHS for LHS
            *p_lhs = zvp_clone_ex (rhs);
        } else {
            // Share a copy
            rhs->refcount++;
            *p_lhs = rhs;
        }
    }
}
```

Introduction to phc
Challenges to compilation
phc solution: use the C API
Speedup
A Practical Solution for Scripting Language Compilers

Example: printf ($f)
A Practical Solution for Scripting Language Compilers

Outline

1. Introduction to phc
2. Challenges to compilation
3. phc solution: use the C API
4. Speedup

Speedup

Trinity College Dublin
1. Why is experimental evaluation a speedup?
2. That's an interesting result. Shouldn't compilers always be faster!!!
3. PHP's interpreter isn't slowed by interpreter loop. Rather it's the level of dynamicism.
1. each statement is pretty high level

```php
<?php
for ($i = 0; $i < $n; $i++)
    $str = $str . "hello";
?>

<?php
for ($i = 0; $i < $n; $i++)
{
    $T = $str . "hello";
    $str = $T;
}
?>
```
A Practical Solution for Scripting Language Compilers

1. We don't need to know how to fold constants - we just pass it off to PHP's eval
2. PHP implements this
3. Function can't change after first invocation - don't need lookup-cache of inline cache or polymorphic inline cache

```
<?php
...
$T = "5" + true;
...
?>
```

```
<?php
...
$T = 6;
...
?>
```
A Practical Solution for Scripting Language Compilers

Optimization

1. We don't need to know how to fold constants - we just pass it off to PHP's eval
2. PHP implements this
3. Function can change after first invocation - don't need lookup-cache of inline cache or polymorphic inline cache

<?php
    $sum = 0;
    for ($i = 0; $i < 10; $i=$i+1)
    {
        $sum .= "hello";
    }
?>
1. We don't need to know how to fold constants - we just pass it off to PHP's `eval`.
2. PHP implements this.
3. Function can't change after first invocation - don't need lookup-cache of inline cache or polymorphic inline cache.
Optimization

- Constant folding
- Constant pooling
- Function caching
- Pre-hashing

```
// $i = 0;
{
    zval* p_i;
    php_hash_find (LOCAL_ST, "i", 5863374, p_i);
    php_destruct (p_i);
    php_allocate (p_i);
    ZVAL_LONG (*p_i, 0);
}
```
1. We don't need to know how to fold constants - we just pass it off to PHP's eval
2. PHP implements this
3. function cant change after first invocation - don't need lookup-cache of inline cache or polymorphic inline cache

Optimization

- Constant folding
- Constant pooling
- Function caching
- Pre-hashing
- Symbol-table removal

```php
// $i = 0;
{
    php_destruct (local_i);
    php_allocate (local_i);
    ZVAL_LONG (*local_i, 0);
}
```
1. Explain how to read graph
2. Much better than 0.1x
3. C compiler: be 5x faster
4. PHP 40x-70x slower
Scripting languages pose new problems for compilers
Solution: Re-use existing run-time
  Speed-ups of 1.5x
  Future work: Precise optimization required for speed
http://phpcompiler.org