

Compiling and Optimizing Scripting Languages

Paul Biggar and David Gregg

Department of Computer Science and Statistics
Trinity College Dublin

Facebook, 16th March, 2009

Motivation

- User needs web page in 0.5 seconds
 - Execution time
 - DB access
 - Network latency
 - Browser rendering
- Easier maintainance
- What if execution was:
 - 2x as fast?
 - 10x as fast?

Outline

- 1 Introduction to phc
- 2 Current state of phc
 - Challenges to compilation?
 - phc solution: use the C API
 - Speedup
- 3 Next for phc - Analysis and Optimization
 - Simple Optimizations
 - Advanced Optimizations
- 4 Experiences with PHP

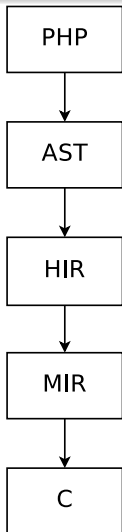
Outline

- 1 Introduction to phc
- 2 Current state of phc
 - Challenges to compilation?
 - phc solution: use the C API
 - Speedup
- 3 Next for phc - Analysis and Optimization
 - Simple Optimizations
 - Advanced Optimizations
- 4 Experiences with PHP

phc

- `http://phpcompiler.org`
- Ahead-of-time compiler for PHP
- Edsko de Vries, John Gilbert, Paul Biggar
- BSD license
- Latest release: 0.2.0.3 - compiles non-OO
- svn trunk: compiles most OO

Structure of **phc**



PHP

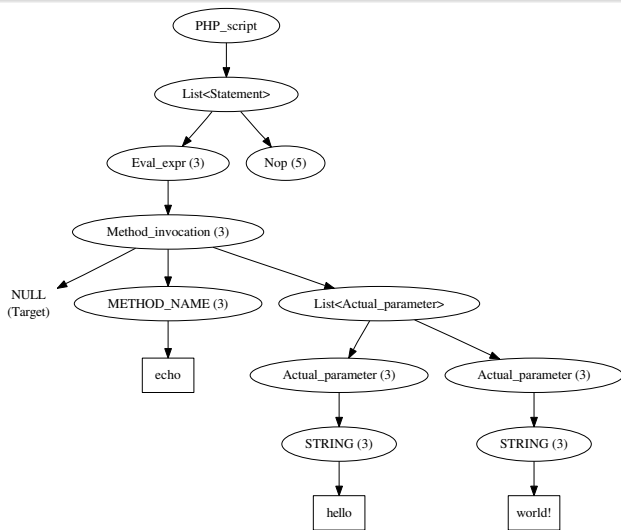


```
<?php
```

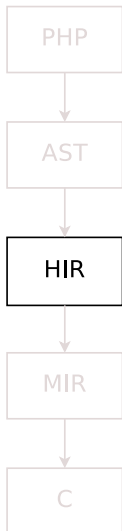
```
echo "hello", "world!";
```

```
?>
```

AST



HIR



```
<?php  
    $x = $a + $b + $c + $d;  
?>
```

```
<?php  
    $TLE0 = ($a + $b);  
    $TLE1 = ($TLE0 + $c);  
    $x = ($TLE1 + $d);  
?>
```

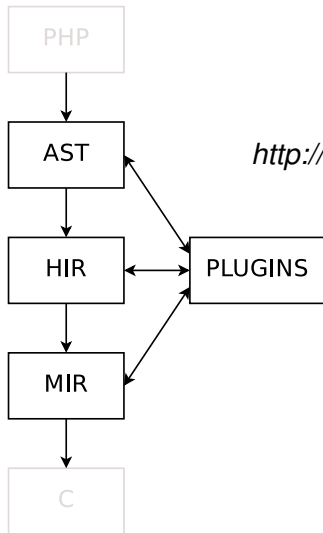
MIR



```
<?php
  while ($cond)
    echo "hello", "world!";
?>
```

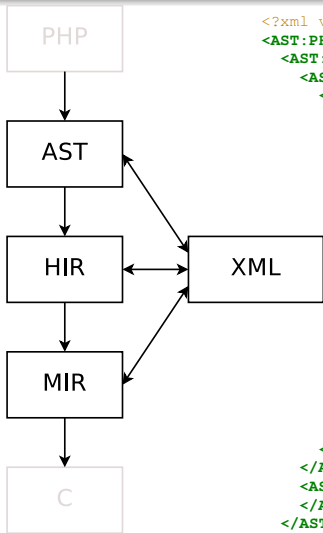
```
<?php
L7:
  $TLE0 = !$cond;
  if ($TLE0) goto L3 else goto L6;
L6:
  print('hello');
  print('world!');
  goto L7;
L3:
?>
```

Plugins



<http://phpcompiler.org/doc/latest/devmanual.html>

XML



```
<?xml version="1.0"?>
<AST:PHP_script xmlns:AST="http://www.phpcompiler.org/phc-1.1">
  <AST:Statement_list>
    <AST:Eval_expr>
      <AST:Method_invocation>
        <AST:Target xsi:nil="true" />
        <AST:METHOD_NAME>
          <value>echo</value>
        </AST:METHOD_NAME>
        <AST:Actual_parameter_list>
          <AST:Actual_parameter>
            <bool><!-- is_ref -->false</bool>
            <AST:STRING>
              <value>hello</value>
            </AST:STRING>
          </AST:Actual_parameter>
          <AST:Actual_parameter>
            <bool><!-- is_ref -->false</bool>
            <AST:STRING>
              <value>world!</value>
            </AST:STRING>
          </AST:Actual_parameter>
        </AST:Actual_parameter_list>
      </AST:Method_invocation>
    </AST:Eval_expr>
    <AST:Nop>
  </AST:Nop>
</AST:Statement_list>
</AST:PHP_script>
```

Outline

- 1 Introduction to phc
- 2 **Current state of phc**
 - Challenges to compilation?
 - phc solution: use the C API
 - Speedup
- 3 Next for phc - Analysis and Optimization
 - Simple Optimizations
 - Advanced Optimizations
- 4 Experiences with PHP

SAC 2009

A Practical Solution for Scripting Language Compilers

Paul Biggar, Edsko de Vries and David Gregg

Department of Computer Science and Statistics
Trinity College Dublin

ACM Symposium on Applied Computing - PL track
12th March, 2009

Sneak peak

- Problem: Scripting languages present “unique” problems (in practice)
- Solution: Re-use as much of the *~~Canonical Reference Implementation~~* as possible.

Outline

- 1 Introduction to phc
- 2 **Current state of phc**
 - Challenges to compilation?
 - phc solution: use the C API
 - Speedup
- 3 Next for phc - Analysis and Optimization
 - Simple Optimizations
 - Advanced Optimizations
- 4 Experiences with PHP

Undefined

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.

D. M. Jones. Forms of language specification: Examples from commonly used computer languages. ISO/IEC JTC1/SC22/OWG/N0121, February 2008.

Batteries included

```

abs()
acos()
acosh()
addslashes()
addslashes()
aggregate()
aggregate_info()
aggregate_methods()
aggregate_methods_by_list()
aggregate_methods_by_regexp()
aggregate_properties()
aggregate_properties_by_list()
aggregate_properties_by_regexp()
aggregation_info()
apache_child_terminate()
apache_get_modules()
apache_get_version()
apache_getenv()
apache_lookup_uri()
apache_note()
apache_request_headers()
apache_reset_timeout()
apache_response_headers()
apache_setenv()
apc_add()
apc_cache_info()
apc_clear_cache()
apc_compile_file()
apc_define_constants()
apc_delete()
apc_fetch()

apc_load_constants()
apc_sma_info()
apc_store()
apd_breakpoint()
apd_callstack()
apd_clunk()
apd_continue()
apd_croak()
apd_dump_function_table()
apd_dump_persistent_resources()
apd_dump_regular_resources()
apd_echo()
apd_get_active_symbols()
apd_set_pprof_trace()
apd_set_session()
apd_set_session_trace()
apd_set_socket_session_trace()
array()
array_change_key_case()
array_chunk()
array_combine()
array_count_values()
array_diff()
array_diff_assoc()
array_diff_key()
array_diff_uassoc()
array_diff_ukey()
array_fill()
array_fill_keys()
array_filter()
array_flip()

array_intersect()
array_intersect_assoc()
array_intersect_key()
array_intersect_uassoc()
array_intersect_ukey()
array_key_exists()
array_keys()
array_map()
array_merge()
array_merge_recursive()
array_multisort()
array_pad()
array_pop()
array_product()
array_push()
array_rand()
array_reduce()
array_reverse()
array_search()
array_shift()
array_slice()
array_splice()
array_sum()
array_udiff()
array_udiff_assoc()
array_udiff_uassoc()
array_uintersect()
array_uintersect_assoc()
array_uintersect_uassoc()
array_unique()
array_unshift()

array_values()
array_walk()
array_walk_recursive()
ArrayIterator::current()
ArrayIterator::key()
ArrayIterator::next()
ArrayIterator::rewind()
ArrayIterator::seek()
ArrayIterator::valid()
ArrayObject::__construct()
ArrayObject::append()
ArrayObject::count()
ArrayObject::getIterator()
ArrayObject::offsetExists()
ArrayObject::offsetGet()
ArrayObject::offsetSet()
ArrayObject::offsetUnset()
arsort()
ascii2ebcdic()
asin()
asinh()
assert()
aspell_check()
aspell_check_raw()
aspell_new()
aspell_suggest()
assert()
assert_options()
atan()
atan2()
atanh()

```

4

Jeff Atwood, Coding Horror, May 20th, 2008

<http://www.codinghorror.com/blog/archives/001119.html>

Change between releases

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

PHP 5.2.1 (32-bit)

int(2147483647)

PHP 5.2.3 (32-bit)

float(2678128395)

Run-time code generation

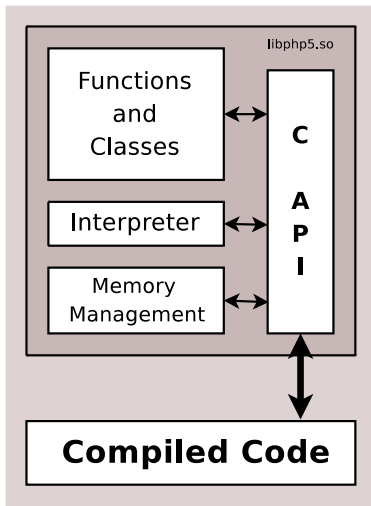
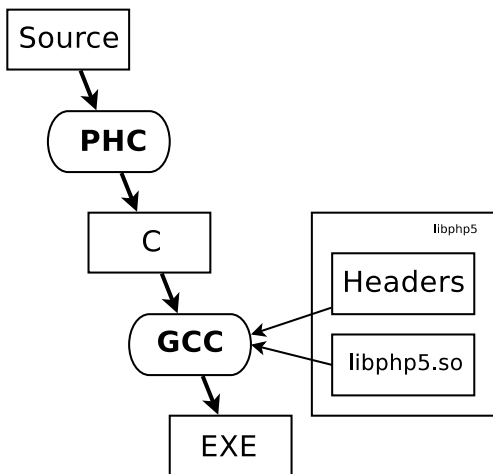
```
<?php
    eval ($argv[1]);
?>
```

```
<?php
    include ("mylib.php");
    ...
    include ("plugin.php");
    ...
?>
```

Outline

- 1 Introduction to phc
- 2 Current state of phc
 - Challenges to compilation?
 - **phc solution: use the C API**
 - Speedup
- 3 Next for phc - Analysis and Optimization
 - Simple Optimizations
 - Advanced Optimizations
- 4 Experiences with PHP

Use C API



More detail

PHP	zval
Python	PyObject
Ruby	VALUE
Lua	TValue

H. Muhammad and R. Ierusalimschy. C APIs in extension and extensible languages. *Journal of Universal Computer Science*, 13(6):839–853, 2007.

Simple listings: \$i = 0

```
// $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

```
// $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;
      }
    }
  }
}
```

Example: printf (\$f)

```
int main(int argc, char *argv[]) {
    printf("Hello, World!\n");
    return 0;
}

```

Applicability

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

Applicability

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

- **Except specification**
 - Lua
 - Python

Applicability

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

- Except specification
 - Lua
 - Python

- **Not at all**
 - Javascript

Outline

- 1 Introduction to phc
- 2 **Current state of phc**
 - Challenges to compilation?
 - phc solution: use the C API
 - **Speedup**
- 3 Next for phc - Analysis and Optimization
 - Simple Optimizations
 - Advanced Optimizations
- 4 Experiences with PHP

Original Speed-up

0.1x

(10 times slower than the PHP interpreter)

The problem with copies

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

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

Optimization

- Constant folding

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

```
<?php
...
$I = 6;
...
?>
```

Optimization

- Constant folding
- **Constant pooling**

```
<?php
    $sum = 0;
    for ($i = 0; $i < 10; $i=$i+1)
    {
        $sum .= "hello";
    }
?>
```

Optimization

- Constant folding
- Constant pooling
- **Function caching**

```
// printf ($f);  
static php_fcall_info printf_info;  
{  
    php_fcall_info_init ("printf", &printf_info);  
  
    php_hash_find (  
        LOCAL_ST, "f", 5863275, &printf_info.params);  
  
    php_call_function (&printf_info);  
}
```

Optimization

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

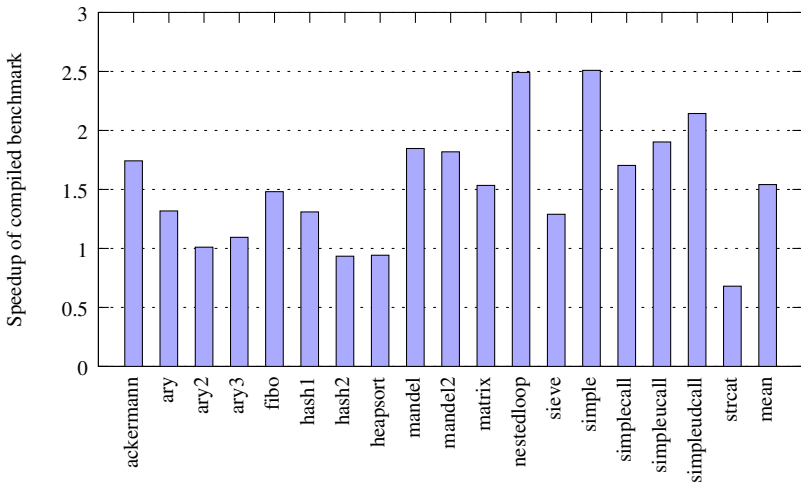
```
// $i = 0;  
{  
    zval* p_i;  
    php_hash_find (LOCAL_ST, "i", 5863374, p_i);  
    php_destruct (p_i);  
    php_allocate (p_i);  
    ZVAL_LONG (*p_i, 0);  
}
```

Optimization

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

```
// $i = 0;  
{  
    php_destruct (local_i);  
    php_allocate (local_i);  
    ZVAL_LONG (*local_i, 0);  
}
```

Current speed-up



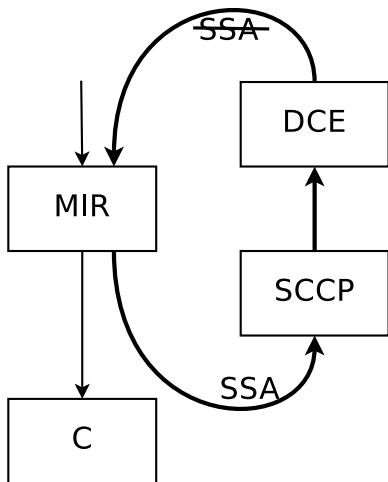
Outline

- 1 Introduction to phc
- 2 Current state of phc
 - Challenges to compilation?
 - phc solution: use the C API
 - Speedup
- 3 Next for phc - Analysis and Optimization
 - Simple Optimizations
 - Advanced Optimizations
- 4 Experiences with PHP

Outline

- 1 Introduction to phc
- 2 Current state of phc
 - Challenges to compilation?
 - phc solution: use the C API
 - Speedup
- 3 Next for phc - Analysis and Optimization
 - **Simple Optimizations**
 - Advanced Optimizations
- 4 Experiences with PHP

Intra-procedural optimizations



- Dead-code elimination
- Sparse-conditional **constant propagation**

Type-inference

```
<?php
```

```
function a ($x, $y)
{
    $str = $x . $y;
    ...
return $str;
}
```

```
?>
```

User-space handlers

- `__toString`
- `__get`
- `__set`
- `__isset`
- `__unset`
- `__sleep`
- `__wake`
- `__call`
- `__callStatic`
- ...

C API handlers

- read_property
- read_dimension
- **get**
- **set**
- cast_object
- has_property
- unset_property
- ...

Unknown types propagate

- local symbol table
- global symbol table
- return values
- reference parameters
- callee parameters

Outline

- 1 Introduction to phc
- 2 Current state of phc
 - Challenges to compilation?
 - phc solution: use the C API
 - Speedup
- 3 Next for phc - Analysis and Optimization
 - Simple Optimizations
 - **Advanced Optimizations**
- 4 Experiences with PHP

Analysis design

- Must model types precisely
 - (Possibly unnamed) fields, arrays, variables and method calls

Analysis design

- Must model types precisely
 - (Possibly unnamed) fields, arrays, variables and method calls
- Uses and definitions incomplete
 - Can't use *def-use chains*
 - Can't use *SSA*

Analysis design

- Must model types precisely
 - (Possibly unnamed) fields, arrays, variables and method calls
- Uses and definitions incomplete
 - Can't use *def-use chains*
 - Can't use *SSA*
- Imprecise callgraph

Algorithm

- Abstract Execution / Interpretation

Algorithm

- Abstract Execution / Interpretation
- Points-to analysis
 - *-sensitive

Algorithm

- Abstract Execution / Interpretation
- Points-to analysis
 - *-sensitive
- Constant-propagation
 - Precision
 - Array-indices/field names
 - Implicit conversions

A. Pioli. Conditional pointer aliasing and constant propagation.
Master's thesis, SUNY at New Paltz, 1999.

Algorithm

- Abstract Execution / Interpretation
- Points-to analysis
 - *-sensitive
- Constant-propagation
 - Precision
 - Array-indices/field names
 - Implicit conversions
- Type-inference
 - Virtual calls
 - Function annotations

Complex cases

- Hashtables
- Implicit conversions
- Variable-variables
- \$GLOBALS
- Static includes
- \$SESSION
- Compiler temporaries

Interesting thoughts

- Strip off first loop iteration

Interesting thoughts

- Strip off first loop iteration
- Analysis at run-time?

Interesting thoughts

- Strip off first loop iteration
- Analysis at run-time?
- Use string transducer analysis

Sound and Precise Analysis of Web Applications
for Injection Vulnerabilities
Gary Wassermann, Zhendong Su, PLDI'07.

Static approximation of dynamically generated Web pages
Yasuhiko Minamide, WWW 2005

Outline

- 1 Introduction to phc
- 2 Current state of phc
 - Challenges to compilation?
 - phc solution: use the C API
 - Speedup
- 3 Next for phc - Analysis and Optimization
 - Simple Optimizations
 - Advanced Optimizations
- 4 Experiences with PHP

Opinions and conjecture

Opinions and conjecture

Opinions and conjecture

Opinions and conjecture

- Language Problems

Opinions and conjecture

Opinions and conjecture

- Language Problems
- Implementation problems

Opinions and conjecture

Opinions and conjecture

- Language Problems
- Implementation problems
- Community Problems

Opinions and conjecture

Fixes

- Remove coupling between libraries and interpreter
- Love of the language leads to more tools

Summary

- Re-use existing run-time for language
- Better yet: standardize language, including FFI
- Analysis needs to be precise, and whole-program
- Pessimistic assumptions spread
- Language, implementation and community need to be fixed
 - All related?

Thanks

phc needs contributors

- **contribute:**
`http://phpcompiler.org/contribute.html`
- **mailing list:** `phc-general@phpcompiler.org`
- **slides:** `http://www.cs.tcd.ie/~pbiggar/`
- **contact:** `paul.biggar@gmail.com`

Complex cases

- Hashtables
- Implicit conversions
- Variable-variables
- \$GLOBALS
- Static includes
- \$SESSION
- Compiler temporaries