

Compiling and Optimizing Scripting Languages

Paul Biggar and David Gregg

Department of Computer Science and Statistics
Trinity College Dublin

OSS Bar Camp, 28th March, 2009

Compiling and Optimizing Scripting Languages

Paul Biggar and David Gregg

Department of Computer Science and Statistics
Trinity College Dublin

OSS Bar Camp, 28th March, 2009

Motivation

1. dont have to
obduscate your
code for
performance

- 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?

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

Outline

- 1 Introduction to phc
- 2 Current state of phc
 -
 -
 -
- 3 Next for phc - Analysis and Optimization
 -
 -
- 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

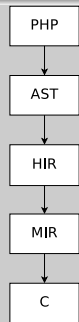
1. BSD licence useful since its easy to extend
2. Well engineered - turns out you dont get a phd for that

- <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

phc

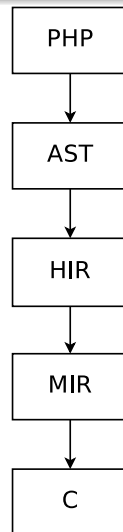
- <http://phpcompiler.org>
- Ahead-of-time compiler for PHP
- Edsko de Vries, John Gilbert, Paul Biggar
- BSD license
- Latest release: 0.2.0.3 - compiles non-OO
- svn trunk: compiles most OO

Structure of phc



1. All can be unparsed to PHP

Structure of phc



PHP

PHP

AST

```
<?php
```

HIR

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

```
?>
```

MIR

C

PHP

PHP

AST

```
<?php
```

HIR

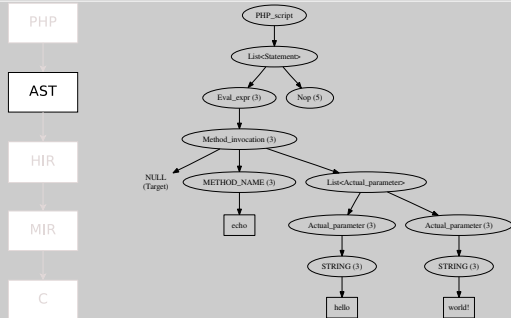
```
    echo "hello", "world!";
```

```
?>
```

MIR

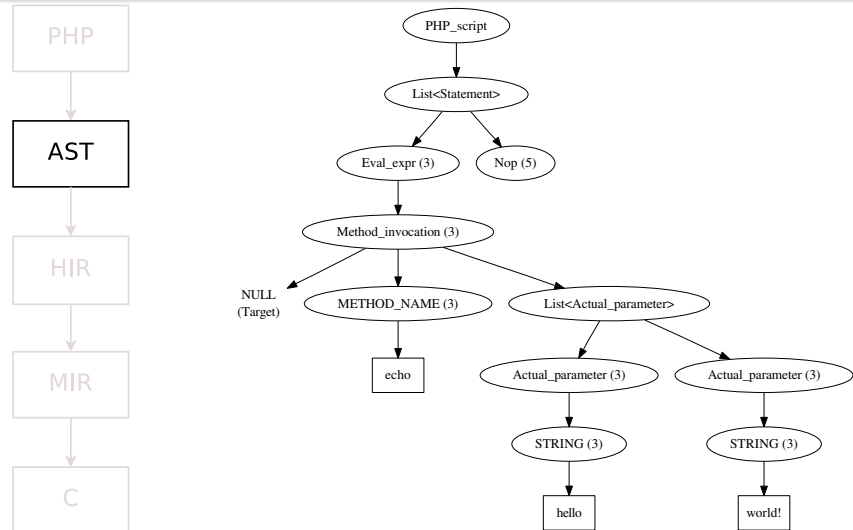
C

AST

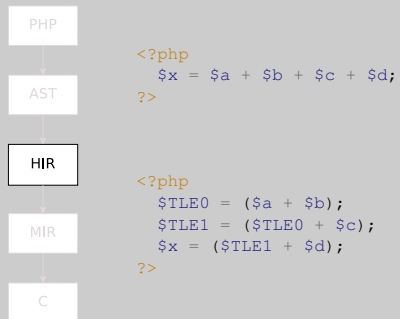


1. better than parse
("concrete syntax")
tree

AST

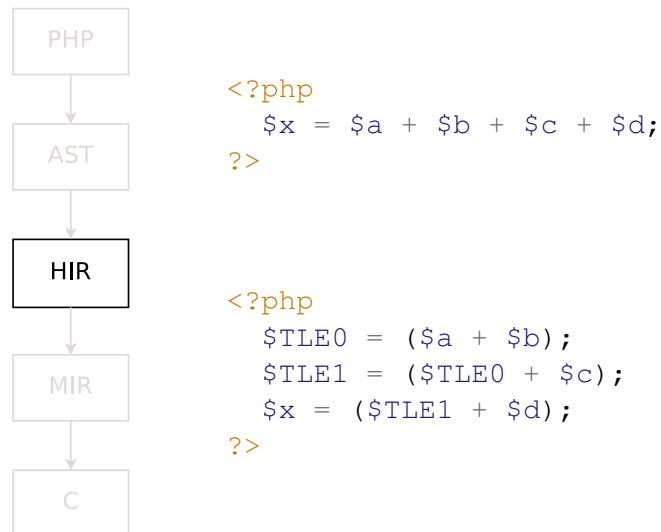


HIR

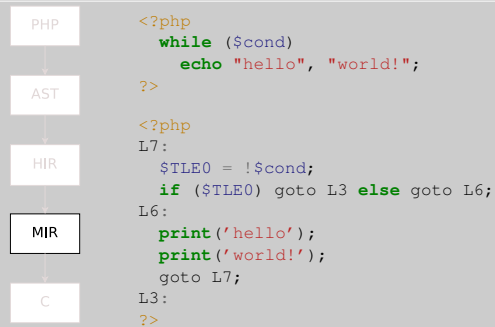


1. 3AC
2. Still PHP

HIR

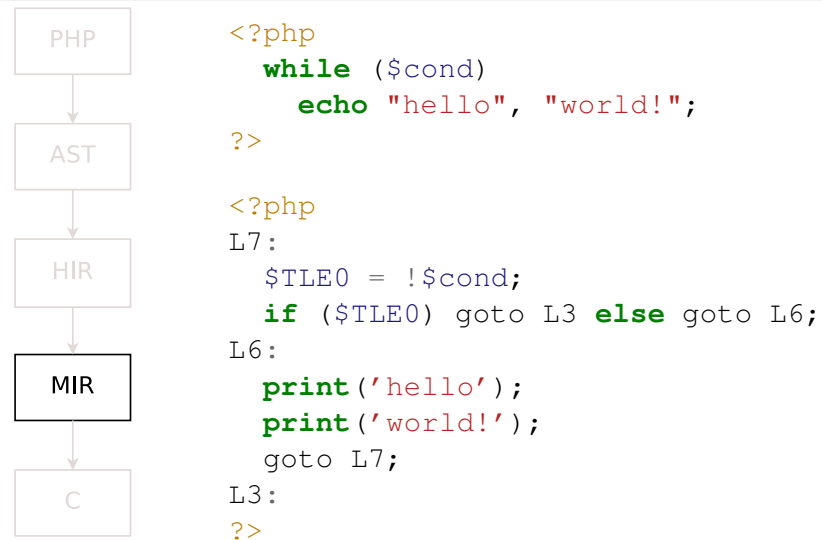


MIR



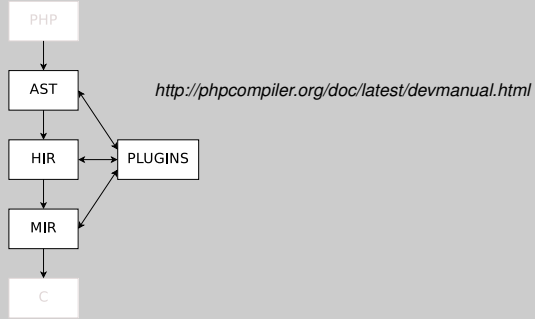
1. Not PHP
2. Gotos

MIR

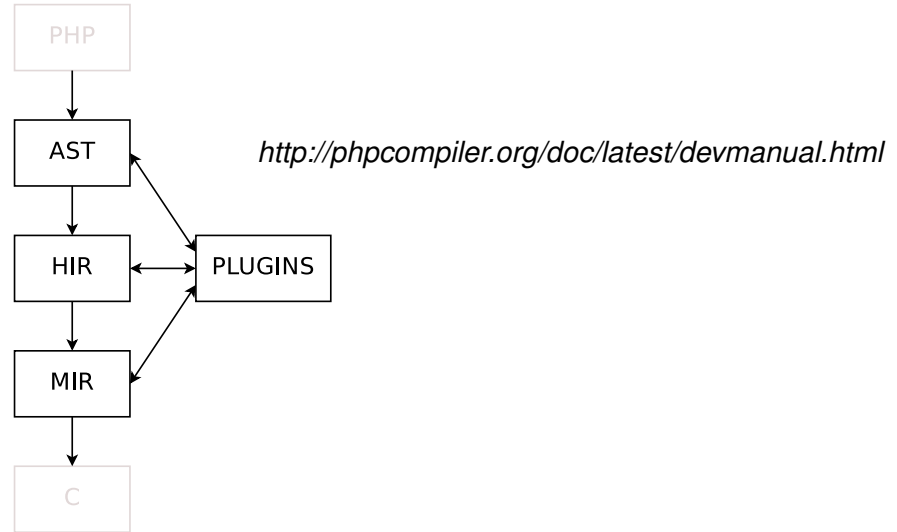


Plugins

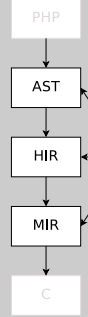
1. Visitor pattern
2. We use it for testing a lot
3. Manual documents it well
- 4.



Plugins



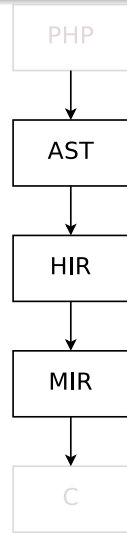
XML



```

<?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>
    
```

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
 -
 -
- 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

1. Correctness
2. Large libraries
3. Odd features
4. No spec

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

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.

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?
 -
 -
- 3 Next for phc - Analysis and Optimization
 -
 -
- 4 Experiences with PHP

Introduction to phc
Current state of phc
Next for phc - Analysis and Optimization
Experiences with PHP

Challenges to compilation?
phc solution: use the C API
Speedup

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.

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

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

```
abs() apc_load_constants() array_intersect() array_values()
acos() apc_sma_info() array_intersect_assoc() array_walk()
acosh() apc_store() array_intersect_key() array_walk_recursive()
addslashes() apd_breakpoint() array_intersect_uassoc() ArrayIterator::current()
addslashes() apd_callstack() array_intersect_ukey() ArrayIterator::key()
aggregate() apd_clunk() array_key_exists() ArrayIterator::next()
aggregate_info() apd_continue() array_keys() ArrayIterator::rewind()
aggregate_methods() apd_croak() array_map() ArrayIterator::seek()
aggregate_methods_by_list() apd_dump_function_table() array_merge() ArrayIterator::valid()
aggregate_methods_by_regexp() apd_dump_persistent_resources() array_merge_recursive() ArrayObject::__construct()
aggregate_properties() apd_dump_regular_resources() array_multisort() ArrayObject::append()
aggregate_properties_by_list() apd_echo() array_pad() ArrayObject::count()
aggregate_properties_by_regexp() apd_get_active_symbols() array_pop() ArrayObject::getIterator()
aggregation_info() apd_set_pprof_trace() array_product() ArrayObject::offsetExists()
apache_child_terminate() apd_set_session() array_push() ArrayObject::offsetGet()
apache_get_modules() apd_set_session_trace() array_rand() ArrayObject::offsetSet()
apache_get_version() apd_set_socket_session_trace() array_reduce() ArrayObject::offsetUnset()
apache_getenv() array() array_reverse() asort()
apache_lookup_uri() apache_change_key_case() array_search() ascii2ebcdic()
apache_note() array_chunk() array_shift() asin()
apache_request_headers() array_combine() array_slice() asinh()
apache_reset_timeout() array_count_values() array_splice() asort()
apache_response_headers() array_diff() array_sum() aspell_check()
apache_setenv() array_diff_assoc() array_udiff() aspell_check_raw()
apc_add() apc_cache_info() array_diff_key() array_udiff_assoc() aspell_new()
apc_clear_cache() apc_clear_cache() array_diff_uassoc() array_udiff_uassoc() aspell_suggest()
apc_compile_file() apc_compile_file() array_diff_ukey() array_uintersect() assert()
apc_define_constants() apc_define_constants() array_fill() array_uintersect_assoc() assert_options()
apc_delete() apc_delete() array_fill_keys() array_uintersect_uassoc() atan()
apc_fetch() apc_fetch() array_flip() array_unshift() atan2()
apc_fetch() apc_fetch() array_flip() array_unshift() atanh()
```

Jeff Atwood, Coding Horror, May 20th, 2008
<http://www.codinghorror.com/blog/archives/001119.html>

Batteries included

```
abs() apc_load_constants() array_intersect() array_values()
acos() apc_sma_info() array_intersect_assoc() array_walk()
acosh() apc_store() array_intersect_key() array_walk_recursive()
addslashes() apd_breakpoint() array_intersect_uassoc() ArrayIterator::current()
addslashes() apd_callstack() array_intersect_ukey() ArrayIterator::key()
aggregate() apd_clunk() array_key_exists() ArrayIterator::next()
aggregate_info() apd_continue() array_keys() ArrayIterator::rewind()
aggregate_methods() apd_croak() array_map() ArrayIterator::seek()
aggregate_methods_by_list() apd_dump_function_table() array_merge() ArrayIterator::valid()
aggregate_methods_by_regexp() apd_dump_persistent_resources() array_merge_recursive() ArrayObject::__construct()
aggregate_properties() apd_dump_regular_resources() array_multisort() ArrayObject::append()
aggregate_properties_by_list() apd_echo() array_pad() ArrayObject::count()
aggregate_properties_by_regexp() apd_get_active_symbols() array_pop() ArrayObject::getIterator()
aggregation_info() apd_set_pprof_trace() array_product() ArrayObject::offsetExists()
apache_child_terminate() apd_set_session() array_push() ArrayObject::offsetGet()
apache_get_modules() apd_set_session_trace() array_rand() ArrayObject::offsetSet()
apache_get_version() apd_set_socket_session_trace() array_reduce() ArrayObject::offsetUnset()
apache_getenv() array() array_reverse() asort()
apache_lookup_uri() apache_change_key_case() array_search() ascii2ebcdic()
apache_note() array_chunk() array_shift() asin()
apache_request_headers() array_combine() array_slice() asinh()
apache_reset_timeout() array_count_values() array_splice() asort()
apache_response_headers() array_diff() array_sum() aspell_check()
apache_setenv() array_diff_assoc() array_udiff() aspell_check_raw()
apc_add() apc_cache_info() array_diff_key() array_udiff_assoc() aspell_new()
apc_clear_cache() apc_clear_cache() array_diff_uassoc() array_udiff_uassoc() aspell_suggest()
apc_compile_file() apc_compile_file() array_diff_ukey() array_uintersect() assert()
apc_define_constants() apc_define_constants() array_fill() array_uintersect_assoc() assert_options()
apc_delete() apc_delete() array_fill_keys() array_uintersect_uassoc() atan()
apc_fetch() apc_fetch() array_flip() array_unshift() atan2()
apc_fetch() apc_fetch() array_flip() array_unshift() atanh()
```

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)

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

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");
    ...
?>

```

Run-time code generation

```

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

```

```

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

```

Outline

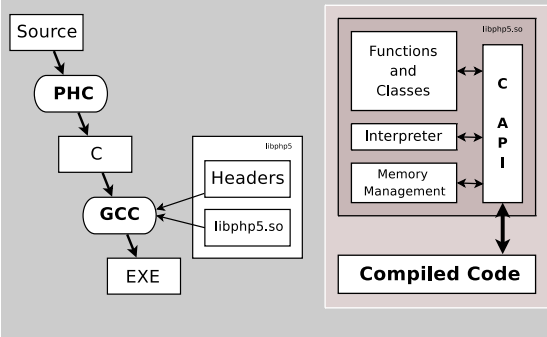
- 1 Introduction to phc
- 2 **Current state of phc**
 - phc solution: use the C API
- 3 Next for phc - Analysis and Optimization
 -
- 4 Experiences with PHP

Outline

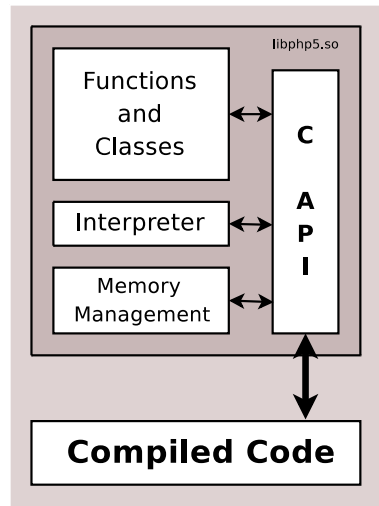
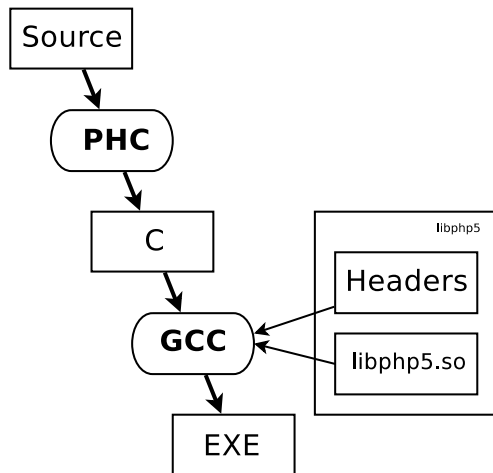
- 1 Introduction to phc
- 2 **Current state of phc**
 - Challenges to compilation?
 - **phc solution: use the C API**
 - Speedup
- 3 Next for phc - Analysis and Optimization
 - Simple Optimizations
 - Advanced Optimizations
- 4 Experiences with PHP

1. RTCG
2. Functions
3. Changes between releases: also use C API at compile-time

Use C API



Use C API



More detail

1. C API is just zval + macros and functions
2. Use (target) PHP's C API at run-time

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.

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);
}
```

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 = 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: \$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;
}
```

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*

Applicability

1. Python used to be bad - aycock quote

- Everything
 - Perl
 - PHP
 - Ruby
 - Tcl – *I think*
- **Except specification**
 - Lua
 - Python

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

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**
 - Speedup
- 3 Next for phc - Analysis and Optimization
 -
- 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

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.

Original Speed-up

0.1x

(10 times slower than the PHP interpreter)

Original Speed-up

0.1x

(10 times slower than the PHP interpreter)

The problem with copies

1. each statement is pretty high level

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

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

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

- We don't need to know how to fold constants - we just pass it off to PHP's eval

<pre><?php ... \$T = "5" + true; ... ?></pre>	<pre><?php ... \$T = 6; ... ?></pre>
---	--

Optimization

- Constant folding

<pre><?php ... \$T = "5" + true; ... ?></pre>	<pre><?php ... \$T = 6; ... ?></pre>
---	--

Optimization

- Constant folding
- **Constant pooling**

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

Optimization

- Constant folding
- **Constant pooling**

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

Optimization

1. PHP implements this
2. function cant change afte first invocation - dont need lookup-cache of inline cache or polymorphic inline cache

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

Optimization

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

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**

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

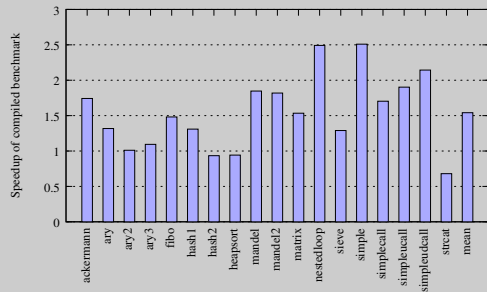
Optimization

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

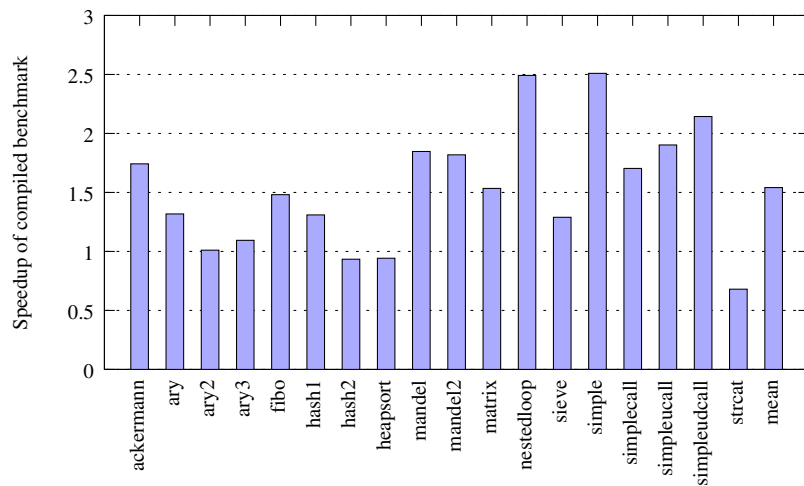
```
// $i = 0;  
{  
  php_destruct (local_i);  
  php_allocate (local_i);  
  ZVAL_LONG (*local_i, 0);  
}
```

Current speed-up

1. Explain how to read graph
2. Much better than 0.1x
3. C compiler: be 5x faster
4. PHP 40x-70x slower



Current speed-up



Outline

- 1 Introduction to phc
- 2 Current state of phc
 -
 -
 -
- 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

Outline

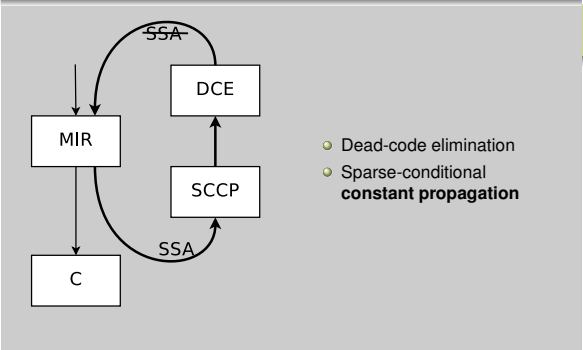
- 1 Introduction to phc
- 2 Current state of phc
 -
 -
 -
- 3 Next for phc - Analysis and Optimization
 - Simple Optimizations
 -
- 4 Experiences with PHP

Outline

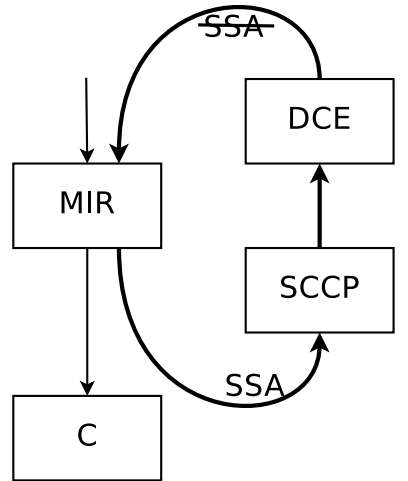
- 1 Introduction to phc
- 2 Current state of phc
 - Challenges to compilation?
 - phc solution: use the C API
 - Speedup
- 3 Next for phc - Analysis and Optimization
 - Simple Optimizations
 - Advanced Optimizations
- 4 Experiences with PHP

Intra-procedural optimizations

1. 2x speedup



Intra-procedural optimizations



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

Type-inference

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

Type-inference

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

User-space handlers

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

User-space handlers

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

C API handlers

1. So previous SSA opts were illegal
2. Complete access to interpreter internals
3. Need accurate use-defs

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

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

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
 -
 -
 -
- 3 Next for phc - Analysis and Optimization
 - 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

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

Analysis design

1. Uses and definitions incomplete - this doesn't use them

- 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*

Analysis design

1. Imprecise callgraph - do it lazily

- 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

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

Algorithm

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

1. flow, context,
object, field

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

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

1. Make polymorphic calls monomorphic
2. Go through each of the problems on the previous slide
3. model types precisely
4. need to model many functions - in contrast to SAC stuff
5. much easier than reimplementing, however

Algorithm

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

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

1. Static-includes optimization needs to be deployment-time
2. hashtables - SAC javascript talk

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

Complex cases

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

Outline

- 1 Introduction to phc
- 2 Current state of phc
 -
 -
 -
- 3 Next for phc - Analysis and Optimization
 -
 -
- 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

Opinions and conjecture

1. hope you like PhD work as much as I hope my externs will
2. working with php

Opinions and conjecture

Opinions and conjecture

Opinions and conjecture

1. difficult to get a good mental model, without looking at source - edge cases
2. Then:
3. dont go into it
4. documented all over web

Opinions and conjecture

- Language Problems

Opinions and conjecture

Opinions and conjecture

- Language Problems

1. implementation slow
2. C API prevents fast implementation - tight coupling - unchangable
3. Instead use libs + good FFI + user-code
4. Allows analysis
5. Other implementations
6. Allow experimentation within Zend engine

Opinions and conjecture

Opinions and conjecture

- Language Problems
- Implementation problems

Opinions and conjecture

Opinions and conjecture

- Language Problems
- Implementation problems

1. compared to GCC
2. bikeshedding
3. more code review - possibly even commit-after-review
4. few working on php core - except in APC
5. IBM, roadsend, davis, tuwien, stanford, utrecht, trinity.
However: YARV, Rubinius, Jython, JRuby, LuaJIT, Pypy, Psyco, Shed-skin, unladen-swallow
6. Why? Love of language

Opinions and conjecture

Opinions and conjecture

- Language Problems
- Implementation problems
- Community Problems

Opinions and conjecture

Opinions and conjecture

- Language Problems
- Implementation problems
- Community Problems

Opinions and conjecture

1. php7k - like py3k
2. break BC
3. remove legacy
4. internally consistent

Fixes

- Remove coupling between libraries and interpreter
- Better community interactions:
 - Pre-commit reviews
 - Mailing list moderation
 - Per-area maintainers
- Love of the language leads to more tools

Opinions and conjecture

Fixes

- Remove coupling between libraries and interpreter
- Better community interactions:
 - Pre-commit reviews
 - Mailing list moderation
 - Per-area maintainers
- Love of the language leads to more tools

Summary

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

Summary

- Re-use existing run-time for language
- Better yet: standardize libraries (and 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`

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

1. Static-includes optimization needs to be deployment-time
2. hashtables - SAC javascript talk

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

Complex cases

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