Inter-Producedural Identical Code Folding in GCC

Martin Liška
SUSE Labs

2015/08/08

Overview

Introduction

Motivation

Large applications, heavily utilizing templates and classes, tend to have a lot of functions that are identical with another functions.

- IPA LTO capable optimization pass
- merges semantically equivalent functions and read-only variables
- motivated by ICF in the Gold Linker
- GCC ICF works on different level and optimization are not equal
- enabled with -O2 and higher; and -Os
- first version of patch: 11/2013
- merged on 10/2014 and being part of GCC 5.1.0
- has caused 44 PRs!

Comparison with ICF in the Gold linker

- Gold works on assembly language level, while GCC operates on GIMPLE level
- Gold can just save size, while GCC can also achieve speed improvement (tramp3d)
- implementation in GCC corresponds to `-icf=safe`
- Gold requires `-ffunction-sections`
- GCC can also merge read-only variables

The Gold Linker vs. GCC

```
int foo(void) { return 3; }
int bar(void) { return 3; }
int main()
{
  int (*a) (void) = &foo;
  int (*b) (void) = &bar;
  return a == b ? 1 : 0;
}
```

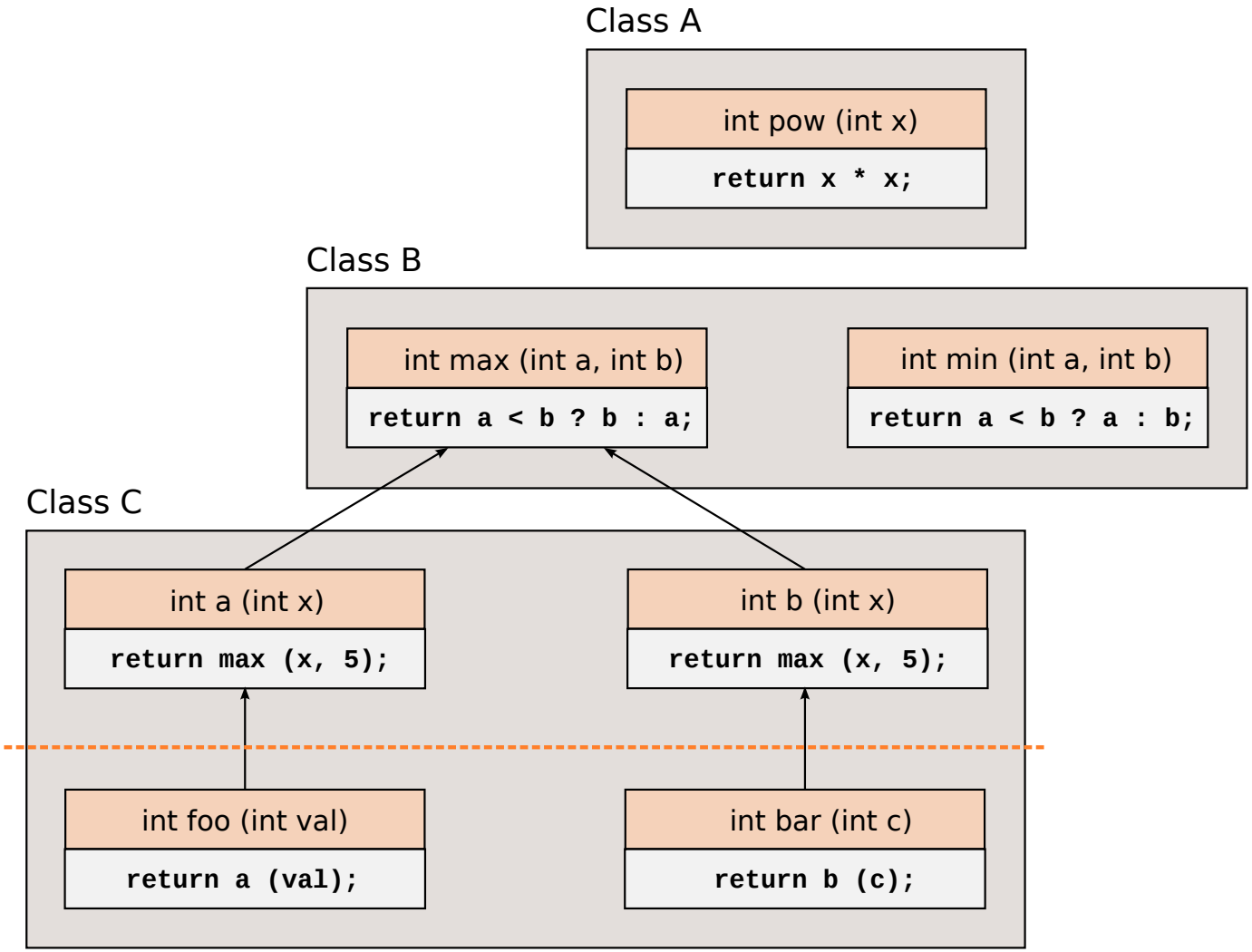
```
$ gcc /tmp/a.c -Wl,--icf=all -Wl,--print-icf-sections
-ffunction-sections 2>&1 | grep foo
ICF folding section '.text.bar' in file '/tmp/x.o' into
'.text.foo' in file '/tmp/y.o'
$ ./a.out && echo $?  => 1
```

Algorithm

Algorithm Description

1. calculate a hash value for each function and read-only variable
2. stream these hash values and load it in WPA phase of LTO compilation
3. filter out removed callgraph nodes
4. identify important references (taken addresses & interposables)
5. enhance hash value
6. build congruence classes based on these hash values

Algorithm Description (congruence classes)



Algorithm Description (continued)

8. subdivide classes by fast WPA comparison
9. process congruence reduction (based on Optimistic VN alg.)
10. load function bodies for non-singleton classes
11. subdivide items in classes by statement based comparison
12. process congruence reduction
13. merge non-singleton classes

Algorithm: Ways of Merging

Read-only variables

1. alias creation
   - if target supports creation of aliases
   - if address not matters or `-fmerge-all-constants`
   - if not discardable and in a same COMDAT group

Functions

1. alias creation
   - if target supports creation of aliases
   - ... many more conditions must be met
2. wrapper creation
   - if e.g. a function has taken address
   - original function body is replaced by a thunk
3. function redirection
   - local calls to the function are redirected
   - the aliasing function can be potentially removed

Comparison of Pair of Functions

Example: source code

```
int foo (long int a, float b)
{
    float c = a + b;
    if (c == b)
        return 1;
    int d = a + 3;
    return d;
}
```

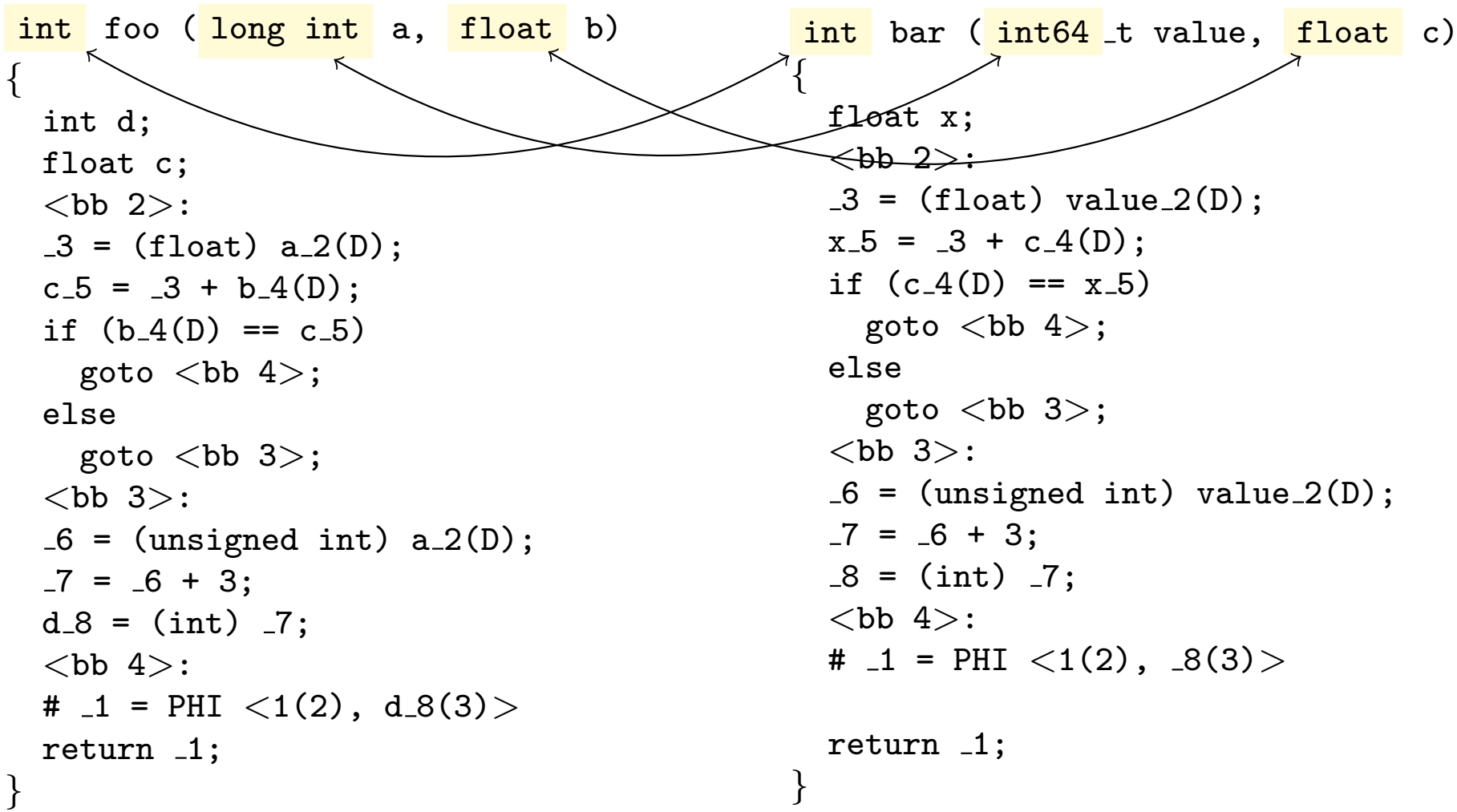
```
int bar (int64_t value, float c)
{
    float x = value + c;
    if (x == c)
        return 1;

    return value + 3;
}
```

Example: GIMPLE Representation

```
int foo (long int a, float b)
{
  int d;
  float c;
  <bb 2>:
  _3 = (float) a_2(D);
  c_5 = _3 + b_4(D);
  if (b_4(D) == c_5)
    goto <bb 4>;
  else
    goto <bb 3>;
  <bb 3>:
  _6 = (unsigned int) a_2(D);
  _7 = _6 + 3;
  d_8 = (int) _7;
  <bb 4>:
  # _1 = PHI <1(2), d_8(3)>
  return _1;
}
```

```
int bar (int64_t value, float c)
{
  float x;
  <bb 2>:
  _3 = (float) value_2(D);
  x_5 = _3 + c_4(D);
  if (c_4(D) == x_5)
    goto <bb 4>;
  else
    goto <bb 3>;
  <bb 3>:
  _6 = (unsigned int) value_2(D);
  _7 = _6 + 3;
  _8 = (int) _7;
  <bb 4>:
  # _1 = PHI <1(2), _8(3)>
  return _1;
}
```

Example: Function Types Comparison



```
int foo (long int a, float b)
{
  int d;
  float c;
  <bb 2>:
  _3 = (float) a_2(D);
  c_5 = _3 + b_4(D);
  if (b_4(D) == c_5)
    goto <bb 4>;
  else
    goto <bb 3>;
  <bb 3>:
  _6 = (unsigned int) a_2(D);
  _7 = _6 + 3;
  d_8 = (int) _7;
  <bb 4>:
  # _1 = PHI <1(2), d_8(3)>
  return _1;
}
```

```
int bar (int64_t value, float c)
{
  float x;
  <bb 2>:
  _3 = (float) value_2(D);
  x_5 = _3 + c_4(D);
  if (c_4(D) == x_5)
    goto <bb 4>;
  else
    goto <bb 3>;
  <bb 3>:
  _6 = (unsigned int) value_2(D);
  _7 = _6 + 3;
  _8 = (int) _7;
  <bb 4>:
  # _1 = PHI <1(2), _8(3)>

  return _1;
}
```

Example: GIMPLE Assignment Comparison

```
int foo (long int a, float b)
{
  int d;
  float c;
  <bb 2>:
  _3 = (float) a_2(D);
  c_5 = _3 + b_4(D);
  if (b_4(D) == c_5)
    goto <bb 4>;
  else
    goto <bb 3>;
  <bb 3>:
  _6 = (unsigned int) a_2(D);
  _7 = _6 + 3;
  d_8 = (int) _7;
  <bb 4>:
  # _1 = PHI <1(2), d_8(3)>
  return _1;
}
```

```
int bar (int64_t value, float c)
{
  float x;
  <bb 2>:
  _3 = (float) value_2(D);
  x_5 = _3 + c_4(D);
  if (c_4(D) == x_5)
    goto <bb 4>;
  else
    goto <bb 3>;
  <bb 3>:
  _6 = (unsigned int) value_2(D);
  _7 = _6 + 3;
  _8 = (int) _7;
  <bb 4>:
  # _1 = PHI <1(2), _8(3)>
  return _1;
}
```

Example: GIMPLE Assignment Comparison

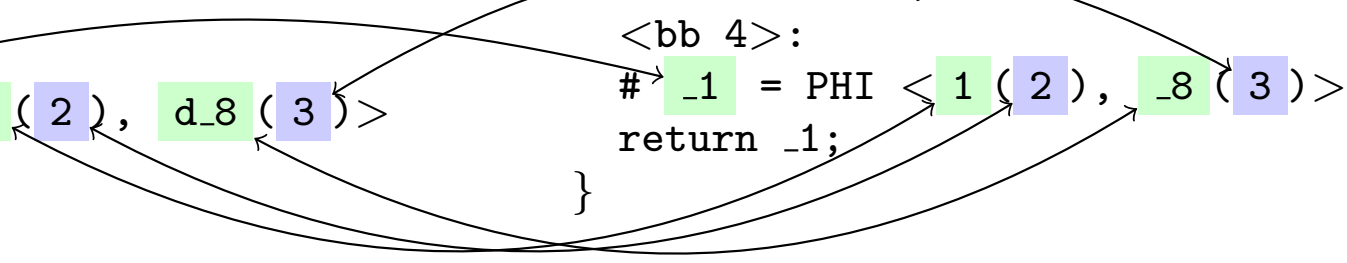
```
int foo (long int a, float b)
{
  int d;
  float c;
  <bb 2>:
  _3 = (float) a_2(D);
  c_5 = _3 + b_4(D);
  if (b_4(D) == c_5)
    goto <bb 4>;
  else
    goto <bb 3>;
  <bb 3>:
  _6 = (unsigned int) a_2(D);
  _7 = _6 + 3;
  d_8 = (int) _7;
  <bb 4>:
  # _1 = PHI <1(2), d_8(3)>
  return _1;
}
```

```
int bar (int64_t value, float c)
{
  float x;
  <bb 2>:
  _3 = (float) value_2(D);
  x_5 = _3 + c_4(D);
  if (c_4(D) == x_5)
    goto <bb 4>;
  else
    goto <bb 3>;
  <bb 3>:
  _6 = (unsigned int) value_2(D);
  _7 = _6 + 3;
  _8 = (int) _7;
  <bb 4>:
  # _1 = PHI <1(2), _8(3)>
  return _1;
}
```

Example: PHI Comparison

```
int foo (long int a, float b)
{
  int d;
  float c;
  < bb 2 >:
  _3 = (float) a_2(D);
  c_5 = _3 + b_4(D);
  if (b_4(D) == c_5)
    goto <bb 4>;
  else
    goto <bb 3>;
  < bb 3 >:
  _6 = (unsigned int) a_2(D);
  _7 = _6 + 3;
  d_8 = (int) _7;
  <bb 4>:
  #  _1  = PHI  < 1 ( 2 ),  d_8 ( 3 )>
  return _1;
}
```

```
int bar (int64_t value, float c)
{
  float x;
  < bb 2 >:
  _3 = (float) value_2(D);
  x_5 = _3 + c_4(D);
  if (c_4(D) == x_5)
    goto <bb 4>;
  else
    goto <bb 3>;
  < bb 3 >:
  _6 = (unsigned int) value_2(D);
  _7 = _6 + 3;
  _8 = (int) _7;
  <bb 4>:
  #  _1  = PHI  < 1 ( 2 ),  _8 ( 3 )>
  return _1;
}
```

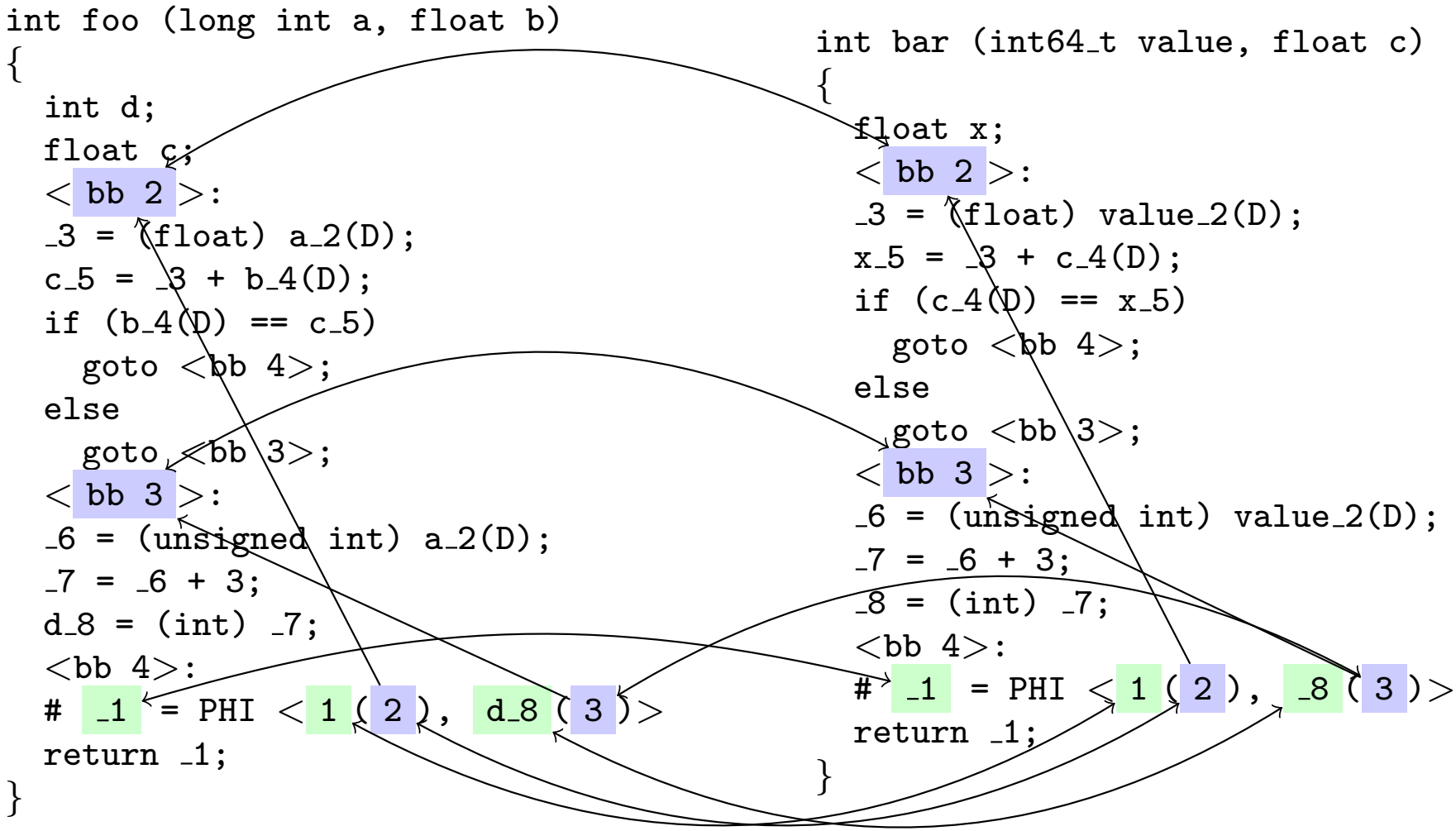


Example: PHI Comparison

```
int foo (long int a, float b)
{
  int d;
  float c;
  < bb 2 >:
  _3 = (float) a_2(D);
  c_5 = _3 + b_4(D);
  if (b_4(D) == c_5)
    goto <bb 4>;
  else
    goto <bb 3>;
  < bb 3 >:
  _6 = (unsigned int) a_2(D);
  _7 = _6 + 3;
  d_8 = (int) _7;
  <bb 4>:
  #  _1  = PHI < 1 ( 2 ),  d_8 ( 3 )>
  return _1;
}
```

```
int bar (int64_t value, float c)
{
  float x;
  < bb 2 >:
  _3 = (float) value_2(D);
  x_5 = _3 + c_4(D);
  if (c_4(D) == x_5)
    goto <bb 4>;
  else
    goto <bb 3>;
  < bb 3 >:
  _6 = (unsigned int) value_2(D);
  _7 = _6 + 3;
  _8 = (int) _7;
  <bb 4>:
  #  _1  = PHI < 1 ( 2 ),  _8 ( 3 )>
  return _1;
}
```

Example: PHI Comparison



Example: A Merged Function

```
foo (long int a, float b)
{
  int retval.2;
  <bb 2>:
  retval.2_3 = bar (a_1(D), b_2(D)); [tail call]
  return retval.2_3;
}
```

```
foo:
.LFB3:
.cfi_startproc
jmp bar
.cfi_endproc
```

Results

Firefox: Results

Lines of code: 19 M
Source files: 11 K header, 10 K source code
Symbol count: 274 344
Execution time in WPA: 4.48 s (5%)

Configuration	code	data	dyn.	EH	TOTAL	**%**
5.1 w/o ICF	43.05	20.07	8.97	8.20	80.35	**100.00**
5.1	41.71	20.00	8.86	7.81	78.48	**97.64**
5.1 w/ safe GOLD	42.90	20.05	8.81	8.20	80.01	**99.57**
6.0 experimental	41.18	20.02	8.77	7.75	77.78	**96.80**
6.0 w/ safe GOLD	41.18	20.02	8.77	7.75	77.78	**96.80**
6.0 w/ all GOLD	41.16	20.02	8.77	7.75	77.76	**96.78**

Table: Binary size (in MB)

Firefox: GCC 6.0 Dump File Analysis

```
Item count: 274344
Congruent classes before: 236680, after: 237338
Average class size before: 1.16, after: 1.16
Average non-singular class size: 6.44, count: 6803
Equal symbols: 37006
Fraction of visited symbols: 13.49%
```

Unified symbols:

```
  542 Unified; Variable alias has been created.
 5724 Unified; Wrapper has been created.
25619 Unified; Function alias has been created.
```

Not unified symbols:

```
   35 Not unifying; can not produce local alias.
 2126 Not unifying; address of original and alias may be compared.
 2957 Not unifying; DECL_NO_INLINE_WARNING mismatch.
```

Chrome: Results

Lines of code: 21 M
Source files: 35 K header, 19 K source code
Symbol count: 487 213
Execution time in WPA: 10.60 s (3%)
Equal symbols: 82 866 (71 681 really unified)

Configuration	code	data	dyn.	EH	TOTAL	**%**
5.1 w/o ICF	66.06	13.35	12.92	10.30	102.68	**100.00**
5.1	61.51	13.22	12.00	10.12	96.90	**94.37**
6.0 experimental	61.23	13.14	11.99	9.92	96.32	**93.81**

Table: Binary size (in MB)

Announcement

LLVMdev: With the switch to clang, binary size dropped about 8% while performance mostly stayed the same.

Chromium: Histogram of Congruence Classes

# symbols	hash based	WPA based	reference based	statement based
1	66.49 %	68.33 %	76.80 %	79.44 %
2	4.69 %	4.24 %	3.61 %	3.34 %
3	1.59 %	1.50 %	1.35 %	1.30 %
4	1.13 %	1.10 %	0.94 %	0.87 %
5	0.78 %	0.73 %	0.60 %	0.58 %
6-9	1.89 %	1.82 %	1.27 %	1.17 %
10-99	8.33 %	8.26 %	6.08 %	5.43 %
100-999	10.70 %	10.31 %	7.21 %	5.74 %
1000-9999	4.45 %	3.76 %	2.14 %	2.14 %

Table: Distribution of congruence classes

Future Work

Future work

- tree-tail-merge pass: replacement of VN with ICF engine
- IPA ICF: relax comparison in compare_operand (GCC 6.0 experimental)
- introduce `-fmerge-all-functions` (equivalent to `-icf=all`)
- clean-up and generalization of `compare_operand`
- sort symbols in a congruence class to maximize unification
- implement at least basic DWARF support
- isolate incremental hashing

Thank you!

Questions?