# **Safe ICF** - Pointer Safe and Unwinding Aware Identical Code Folding in gold

Sriraman Tallam, Cary Coutant, Ian Lance Taylor, David Li, and Chris Demetriou

Google™

# Outline

- Motivation for Identical Code Folding (ICF)
- How identical functions are detected?
- Enforcing safety in the presence of pointer comparisons
- Making binaries with ICF debuggable
- Experimental Results
- Future Work

Google™

# Motivation

- C++ applications using templates tend to have duplicate code
- Upto 10% of functions are duplicates

```
template <typename T>
class Foo
{
  ...
  T element;
public:
  ...
  T getElement ()
  {
    return element;
  }
  ...
};
```

```
int main ()
{
  ...
  Foo<int *> p;
  Foo<float *> q;
  Foo<void *> r;
  ...
}
```

## Identical Functions

Foo<float*>::getElement()
Foo<int*>::getElement()
Foo<void*>::getElement()

Google™

# What is ICF?

- Implemented in the gold linker
- Detects and merges functions with identical object code

```
int foo ()
{
  return 2;
}

int bar ()
{
  // This function returns 2.
  return 2;
}

int main()
{
  return foo() + bar();
}
```

```
000000000040038c <_Z3foov>:
  push  %rbp
  mov   %rsp,%rbp
  mov   $0x2,%eax
  leaveq
  retq

0000000000400397 <_Z3barv>:
  push  %rbp
  mov   %rsp,%rbp
  mov   $0x2,%eax
  leaveq
  retq

00000000004003a2 <main>:
  callq  40038c <_Z3foov>
  callq  400397 <_Z3barv>
```

```
000000000040034c <_Z3barv>:
  push   %rbp
  mov    %rsp,%rbp
  mov    $0x2,%eax
  leaveq
  retq

0000000000400357 <main>:
...
  callq  40034c <_Z3barv>
...
  callq  40034c <_Z3barv>

Output of nm :
40034c T _Z3barv
40034c T _Z3foov
```

Google™

# What are Identical Functions?

- Function = Section + Relocations
- Functions foo and bar are identical if and only if:
  - text sections are bit identical, **and**
  - relocations point to sections that are identical

```
int foo ()
{
  return zip ();
}

int bar ()
{
  return zap ();
}
```

```
int zip ()
{
  return 0;
}
```

```
int zap ()
{
  return 0;
}
```

```
0000000000000000 <_Z3foov>:
  55            push   %rbp
  48 89 e5        mov    %rsp,%rbp
  e8 00 00 00 00  callq  9
    R_X86_64_PC32 relocation to zip
  c9            leaveq
  c3            retq


0000000000000000 <_Z3barv>:
  55            push   %rbp
  48 89 e5        mov    %rsp,%rbp
  e8 00 00 00 00  callq  9
    R_X86_64_PC32 relocation to zap
  c9            leaveq
  c3            retq
```

Google™

# Detecting Identical Functions

## Goal : Form groups of identical functions

- Split relocations into two types
  - Variable - point to function sections
  - Constant - point to other sections
- Initialize group id for all functions - two choices
- Replace variable relocations with group id
- Checksum section contents
- Lookup hash to determine new group id
- Repeat till convergence
- Keep one copy from each group
- Remap symbols of discarded copies

Google™

# Pessimistic Initialization

- **Pessimistic** - Assume all functions are unique
- The detection algorithm need not be run to convergence
- Trouble handling Recursion
  - Self recursion - replace with special symbol
  - Mutual recursion - cannot be folded

```
int funcA (int a)
{
  if (a == 1)
    return 1;
  return 1 + funcB(a - 1);
}
```

```
int funcB (int a)
{
  if (a == 1)
    return 1;
  return 1 + funcA(a - 1);
}
```

funcA & funcB are identical

# Optimistic Initialization

- **Optimistic -** All functions are identical
- The detection algorithm has to be run to convergence
  - Experiments show it needs ~6 iterations to converge
- Recursion is handled automatically

# Our Initialization

We chose the pessimistic approach

- Converges in 3 iterations, default is set to 2
- Modified to handle recursive calls
- No mutually recursive calls in our benchmarks

# Preprocessing & Optimizations

- First Iteration
  - Eliminate functions with unique text content
  - Finalize functions with no variable relocations
- After every iteration
  - Keep one function in each group for the next iteration
  - Finalize other functions
- Checksumming algorithm
  - Used crc32. Multi-map to handle identical checksums
  - md5 is good but slower

Google™

# Execution Safety

- Function pointer comparisons of merged functions
  - Affects execution correctness
- Cause for a crash in our benchmark

```
int foo ()
{
  return 0;
}

int bar ()
{
  return 0;
}

int main ()
{
  assert (foo != bar);
}
```

Google™

# Safe ICF

Idea: Do not fold functions whose pointer is accessed

- Always fold ctors and dtors
  - Their pointer cannot be accessed
- Relocation type disambiguation to identify function pointer accesses
  - Types usually differ for function call and function pointer access
  - Disregard pointer accesses for vtable purposes
  - Assume the worst when disambiguation is not possible

# Relocation type disambiguation

For x86_64:

- Binaries linked with non-PIE objects
  - Function pointer access is a direct relocation
  - Function call is a PC relative relocation
- Shared Objects, only PIC objects
  - Function pointer access is a GOTPCREL relocation
  - Function call is a PC relative relocation
- Binaries linked with PIE objects
  - Cannot be distinguished
  - Conservatively folds only ctors and dtors

# Unwinding across merged functions

- DWARF extensions proposed to disambiguate PC values
- Examine return pointer (call chain) when PC is in merged function
  - Direct call
    - Direct call table maps call-site to debug info
  - Virtual call
    - Virtual call table maps virtual call-site to vtable slot index
  - Other Indirect call
    - Safe ICF prevents such calls from being folded
    - Otherwise cannot disambiguate
  - Tail call
    - Cannot disambiguate the PC

# Experimental Results

| Benchmark | Text size of binaries in MB | | | | | |
|---|---|---|---|---|---|---|
| | Improvement over Baseline | | | Improvement over Linker GC | | |
| | Orig. | With ICF | With SafeICF | GC | With ICF | With SafeICF |
| Index search | 96.06 | 89.73 (6.59%) | 89.77 (6.55%) | 68.35 | 66.25 (3.07%) | 66.34 (2.95%) |
| Database | 61.91 | 57.59 (6.97%) | 57.72 (6.77%) | 47.10 | 45.39 (3.61%) | 45.38 (3.64%) |
| Ads | 121.02 | 111.99 (7.46%) | 112.14 (7.34%) | 85.44 | 82.53 (3.40%) | 82.46 (3.49%) |
| Image Processing - I | 77.84 | 73.96 (4.99%) | 74.09 (4.82%) | 55.67 | 54.33 (2.41%) | 54.38 (2.31%) |
| Image Processing - II | 43.72 | 41.14 (5.89%) | 41.19 (5.78%) | 26.93 | 26.16 (2.87%) | 26.18 (2.81%) |
| File system | 21.11 | 19.79 (6.23%) | 19.83 (6.04%) | 15.47 | 14.99 (3.12%) | 14.99 (3.11%) |
| Web search | 106.61 | 99.79 (6.39%) | 99.85 (6.34%) | 80.86 | 78.31 (3.15%) | 78.46 (2.96%) |
| NLP | 145.41 | 137.01 (5.77%) | 137.34 (5.55%) | 115.31 | 112.13 (2.76%) | 112.35 (2.57%) |
| Text detection | 25.10 | 23.67 (5.68%) | 23.71 (5.55%) | 16.74 | 16.29 (2.72%) | 16.31 (2.58%) |
| Serialization | 60.13 | 55.46 (7.76%) | 55.49 (7.72%) | 42.15 | 40.36 (4.24%) | 40.36 (4.25%) |
| Clustering | 42.49 | 40.39 (4.95%) | 40.44 (4.83%) | 30.12 | 29.45 (2.22%) | 29.50 (2.06%) |
| Map reduce | 93.59 | 87.47 (6.54%) | 87.53 (6.47%) | 71.30 | 69.12 (3.06%) | 69.25 (2.87%) |
| Geo. Mean | 64.10 | 60.08 (6.27%) | 60.15 (6.16%) | 46.02 | 44.61 (3.06%) | 44.65 (2.98%) |

- Text size reduction of ~6% with ICF. 3% over linker garbage collection. Sa ICF is as good as regular.
- No observed performance change.

Google™

# Using Safe ICF

- Safe ICF available in the gold linker
- Flags:
  - **--icf=all** (Full ICF)
  - **--icf=safe** (Safe ICF)
  - **--print-icf-sections** (Print folded sections)
  - **--icf-num-iterations=**

Google™

# Future Work

- Folding identical read only data members in progress
  - Gold already merges string constant