Lightweight, Multi-Stage, Compiler-Assisted Application Specialization (LMCAS)

Mohannad Alhanahnah and Somesh Jha
University of Wisconsin-Madison
Demands for Tiny & Specialized Utilities
Tiny Utilities Generated Manually

Manual

BusyBox
Our Goal
Compilers are Powerful

GCC

- fauto-inc-dec
- fbranch-count-reg
- fcombine-back-adjustments
- fcompare-elim
- fcrop-regs
- fdce
- fdefer-pop
- fdelayed-branch
- fdece
- fforward-propagate
- fgls-listugh
- gcons-branch-probability
- -fif-convexion
- -fif-convexion2
- -fnline-functions-called-once
- -fnipa-ndef
- -fnipa-profile
- -fnipa-pure-const
- -fnipa-reference
- -fnipa-reference-addressable
- -fnmerge-constant
- -fnmove-loop-invariants
- -fnmove-loop-stores
- -fnomit-frame-pointer
- -fnreorder-blocks
- -fnshrink-wrap
- -fnshrink-wrap-separate
- -fnsplit-wide-types
- -fssa-backprop
- -fssa-phiopt
- -ftree-bit-cmp
- -ftree-cpp
- -ftree-ch
- -ftree-coalesce-vars
- -ftree-copy-cprop
- -ftree-dce
- -ftree-dominator-opts
- -ftree-dse
- -ftree-fordrop
- -ftree-fre
- -ftree-phiopt
- -ftree-pa
- -ftree-scev-cprop
- -ftree-sink
- -ftree-sls
- -ftree-ssa
- -ftree-tce
- -funit-at-a-time

---

- falign-functions
- falign-jumps
- falign-loops
- falign-labels
- falign-constants
- fcall-saves
- fbounds-checking
- fforce-jumping
- fforce-follow-jumps
- fforce-skip-blocks
- fdelete-null-pointer-checks
- fdevel
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
- fdevel-pretend
- fdevel-unsafe
- fdevel-unwind
- fdevel-const
Disciplined Software Development

Meinicke et al., Exploring differences and commonalities between feature flags and configuration options. ICSE-SEIP'20
LMCAS Pillars

Intuition
Use Case
LMCAS
Q&A
demo

Program

Configuration
Neck – end of config
Main Logic

GCC

Specialized Utilities

https://github.com/Mohannadcse/LMCAS_Demo
Agenda

• Use Case
• Introducing LMCAS
• Questions
• Demo
Survey

• Do you follow Disciplined Software Development?

YES □

NO □
Agenda

• Use Case
  • Introducing LMCAS
  • Questions
• Demo
Use Case (Network Monitoring)
Use Case (Network Monitoring)

Specialization = High Efficiency
Other Use Cases
Agenda

• Use Case
• Introducing LMCAS
• Questions
• Demo
LMCAS Workflow (Partial Evaluation)
LMCAS Workflow - Examples

\[ \text{tcpdump} \quad \text{LMCAS} \quad \text{Specialized tcpdump} \]

\[ \text{wc} \quad \text{LMCAS} \quad \text{Specialized wc} \]

Intuition

Use Case

Q&A

demo
Illustrative Example

- scaled-down version of the `wc` utility
  1. Line count
  2. Char count

```c
struct Flags {
  char count_chars;
  int count_lines;
};
int total_lines = 0;
int total_chars = 0;
int main(int argc, char** argv){
  struct Flags *flag;
  flag = malloc(sizeof(struct Flags));
  flag->count_chars = 0;
  flag->count_lines = 0;
  if (argc >= 2){
    for (int i = 1; i < argc; i++) {
      if (!strcmp(argv[i], "-c")) flag->count_chars = 1;
      if (!strcmp(argv[i], "-l")) flag->count_lines = 1;
    }
  }
  char buffer[1024];
  while (fgets(buffer, 1024, stdin)){
    if (flag->count_chars) total_chars += decodeChar(buffer);
    if (flag->count_lines) total_lines++;
  }
  if (flag->count_chars) printf("#Chars= %d", total_chars);
  if (flag->count_lines) printf("#Lines= %d", total_lines);
}
```
LMCAS Approach

```c
1 struct Flags {
2   char count_chars;
3   int count_lines;
4 };
5 int total_lines = 0;
6 int total_chars = 0;
7 int main(int argc, char** argv)
8 {
9   struct Flags *flag;
10   flag = malloc(sizeof(struct Flags));
11   flag->count_chars = 0;
12   flag->count_lines = 0;
13   if (argc >= 2){
14       for (int i = 1; i < argc; i++) {
15         if (!strcmp(argv[i], "-c") flag->count_chars = 1;
16         if (!strcmp(argv[i], "-l") flag->count_lines = 1;
17       }
18   }
19   char buffer[1024];
20   while (fgets(buffer, 1024, stdin)){
21       if (flag->count_chars) total_chars += decodeChar(buffer);
22       if (flag->count_lines) total_lines++;
23       if (flag->count_chars) printf("#Chars= %d", total_chars);
24       if (flag->count_lines) printf("#Lines= %d", total_lines);
25   }
26 }
```
LMCAS Architecture

Modified KLEE

Partial Interpretation → Partial Evaluation

LLVM IR

Constant Conversion → Multi-Stage Simplification

Specialized Program

 LLVM IR

Modified KLEE

Neck Miner

Neck Identification (currently manual)

Partial State

Supplied Inputs

LLVM Passes

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Scope</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>total_lines</td>
<td>int</td>
<td>Global</td>
<td>0</td>
</tr>
<tr>
<td>total_chars</td>
<td>int</td>
<td>Global</td>
<td>0</td>
</tr>
<tr>
<td>flag-&gt;count_lines</td>
<td>int</td>
<td>Global</td>
<td>1</td>
</tr>
<tr>
<td>flag-&gt;count_chars</td>
<td>char</td>
<td>Local</td>
<td>0</td>
</tr>
</tbody>
</table>
Neck Properties

• Neck should be an articulation point in the CFG:
  • Dominator of main logic nodes
  • Always executed
    • Reachable from the entry
  • Executed once
    • Outside any loop structure
Partial Interpreter

Modified KLEE
Program Partial State

**Configuration Logic**

```c
struct Flags {
    char count_chars;
    int count_lines;
};
int total_lines = 0;
int total_chars = 0;
int main(int argc, char** argv){
    struct Flags *flag;
    flag = malloc(sizeof(struct Flags));
    flag->count_chars = 0;
    flag->count_lines = 0;
    if (argc >= 2){
        for (int i = 1; i < argc; i++) {
            if (strcmp(argv[i], "-c")) flag->count_chars = 1;
            if (strcmp(argv[i], "-l")) flag->count_lines = 1;
        }
    }
    ...}
```

**Global Variables**
- `total_lines`
- `total_chars`
- `flag->count_chars`
- `flag->count_lines`

**Stack Variables**

**Stored as txt files:**
- `globals=gbls.txt`
- `primitive stack=primitiveLocals.txt`
- `struct stack=customizedLocals.txt`
- `ptr to struct=ptrToStructLocals.txt`
- `ptr to primitive=ptrToPrimitiveLocals.txt`
- `string Vars=stringVars.txt`
- `nested structs=nestedStructLocals.txt`
Partial Interpreter Implementation

KLEE 2.1

LLVM 10.0
Compiler-Assisted Optimizations

Constant Conversion
Multi-stage Simplifications
Constant Conversion (CC)

**Configuration Logic**

```c
struct Flags {
    char count_chars;
    int count_lines;
};
int total_lines = 0;
int total_chars = 0;
int main(int argc, char** argv){
    struct Flags *flag;
    flag = malloc(sizeof(struct Flags));
    flag->count_chars = 0;
    flag->count_lines = 0;
    if (argc >= 2){
        for (int i = 1; i < argc; i++) {
            if (!strcmp(argv[i], "-c")) flag->count_chars = 1;
            if (!strcmp(argv[i], "-l")) flag->count_lines = 1;
        }
    }
    char buffer[1024];
    while (fgets(buffer, 1024, stdin)){
        if (count_chars) total_chars += sizeof(buffer);
        if (count_lines) total_lines++;
    }
    if (count_chars) printf("chars = \%d\n", total_chars);
    if (count_lines) printf("Lines = \%d\n", total_lines);
    return 0;
}
```

**Constant Conversion LLVM Pass**

- `arg1` = `gbls.txt`
- `plocal1` = `primitiveLocals.txt`
- `clocal1` = `customizedLocals.txt`
- `ptrStructLocals` = `ptrToStructLocals.txt`
- `ptrToPrimLocals` = `ptrToPrimitiveLocals.txt`
- `stringVars` = `stringVars.txt`
- `nestedStrcts` = `nestedStructLocals.txt`
CC Pre-neck

```c
int main (int argc, char **argv)
{
    struct rm_options x;

    rm_option_init (&x);

    while ((c = getopt_long (argc, argv, "dfirv", long_opts, NULL)) != -1)
    {
        switch (c)
        {
        case 'd':
            x.remove_empty_directories = true;
            break;

        case 'f':
            x.interactive = RMI_NEVER;
            x.ignore_missing_files = true;
            prompt_once = false;
            break;

        case 'i':
            x.interactive = RMI_ALWAYS;
            x.ignore_missing_files = false;
            prompt_once = false;
            break;

        default:
            break;
        }
    }
}
```

```c
static void rm_option_init (struct rm_options *x)
{
    x->ignore_missing_files = false;
    x->interactive = RMI_SOMETIMES;
    x->one_file_system = false;
    x->remove_empty_directories = false;
    x->recursive = false;
    x->root_dev_ino = NULL;
    x->preserve_all_root = false;
    x->stdin_tty = isatty (STDIN_FILENO);
    x->verbose = false;
    x->require_restore_cwd = false;
}
```

**rm GNU Coreutils**

**Specialize API**

rm_option_init
Multi-stage Simplifications

- Constant Propagation
  - Standard LLVM pass

- Simplifying CFG
  - Standard LLVM Pass

- Cleaning up
  - Customized LLVM Pass
Cleaning up

Removing unused functions

- Iterate the Call Graph
- Check the number of users of each function
- Remove the function if number of users is zero
  - function pointer won’t be removed

Removing unused variables in the remained functions

- Local and global

We used users() LLVM API
LMCAS Power

Lightweight approach
- No test cases/heuristics are required

Reduce programs size
- Binary
  - Functions
  - Basic blocks
  - Instructions

Reduce attack surface
- Eliminate known CVEs
  - Reduce gadgets

Correct functionality

Fully automated
- Except the neck
- Ongoing work

Supports Command-line programs
Agenda

• Use Case
• Introducing LMCAS
• Questions
• Demo
LMCAS - Summary

- Use Case

- LMCAS

- Partial Evaluation

- Partial Interpretation

- Constant Conversion

- Multi-Stage Simplification

- Specialized Program

- Supplied Inputs

- Neck Miner
Agenda

• Use Case
• Introducing LMCAS
• Questions
• Demo
Demo
Roadmap

Two debloating scenarios

See size reduction statistics

See gadgets reduction statistics

Verify the functionality of the debloated program
Debloating Scenarios

Demo Scenarios

Illustrative Example “wc”
- Count # lines and chars

Networking “tcpdump”
- Monitor loopback interface
- Capture only 5 packets

LMCAS Docker and programs used in the demo are available at:
https://github.com/Mohannadcse/LMCAS_Demo

18 Programs
Specializing GNU wc
Specializing tcpdump
Summary

LMCAS specializes programs

- Compiler Optimizations
- Disciplined Implementation

LMCAS Benefits

- Lightweight approach
- Reducing attack surface
- Reducing program size

Ongoing project

- Automating the neck identification
- Leveraging data flow analysis
- Covering more real-world use cases
More Info

GitHub  https://github.com/Mohannadcsel/LMCAS_Demo


Email  mohannad@cs.wisc.edu

Twitter  Malhanahnah