YARPGen: A Compiler Fuzzer for Loop Optimizations and Data-Parallel Languages

Vsevolod Livinskii, University of Utah

Dmitry Babokin, Intel Corporation

John Regehr, University of Utah

November 9th, 2022

Summary of Found Bugs

120 completely new errors in total

40% are wrong code bugs

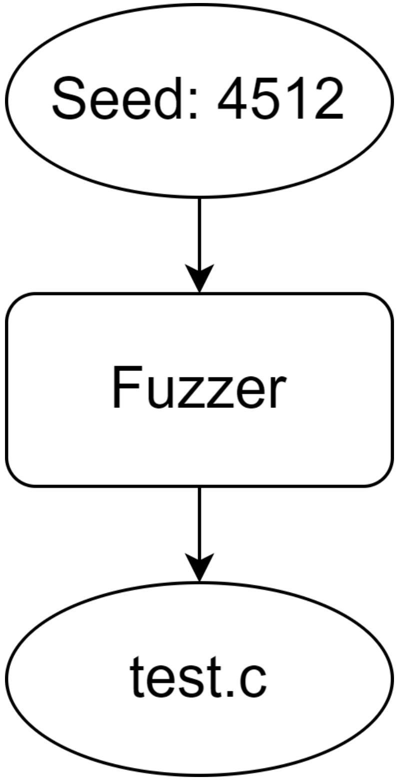
- 27 bugs in LLVM
- 61 bugs in GCC
- 12 bugs in ISPC
- 16 bugs in the DPC++
- 2 bugs in SDE
- 2 bugs in Alive2

YARPGen Features

- Detect wrong code bugs
  - Avoid Undefined Behavior statically
- Target optimizations explicitly
- Easily extensible for C-family languages
  - Including compilers for emerging languages
- Easy to use

Fuzzing Approaches

Generative

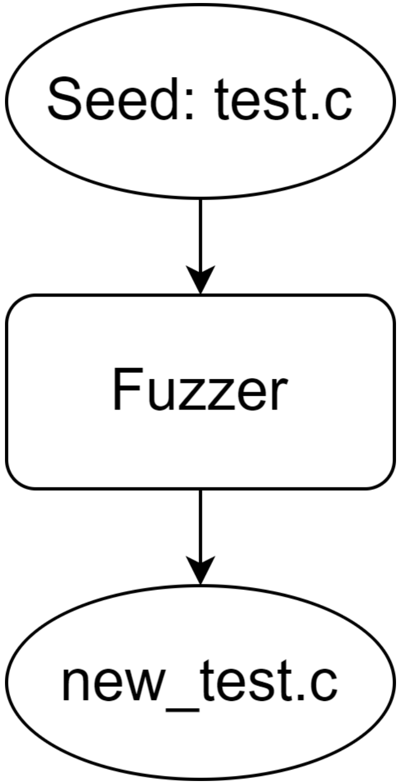


Seed: 4512

Fuzzer

test.c

Mutation-based



Seed: test.c

Fuzzer

new_test.c

YARPGen Features

- Detect wrong code bugs
    - Avoid Undefined Behavior statically
- Target optimizations explicitly
- Easily extensible for C-family languages
    - Including compilers for emerging languages
- Easy to use

Undefined Behavior (UB)

```
# include <stdio.h>

int main () {
    int x = 1;
    x = x++ + ++x;
    printf ("%d\n", x);
    return 0;
}
```

Who is wrong?

```
>$ icc test.cpp && ./a.out
5
>$ clang++ test.cpp && ./a.out
4
```

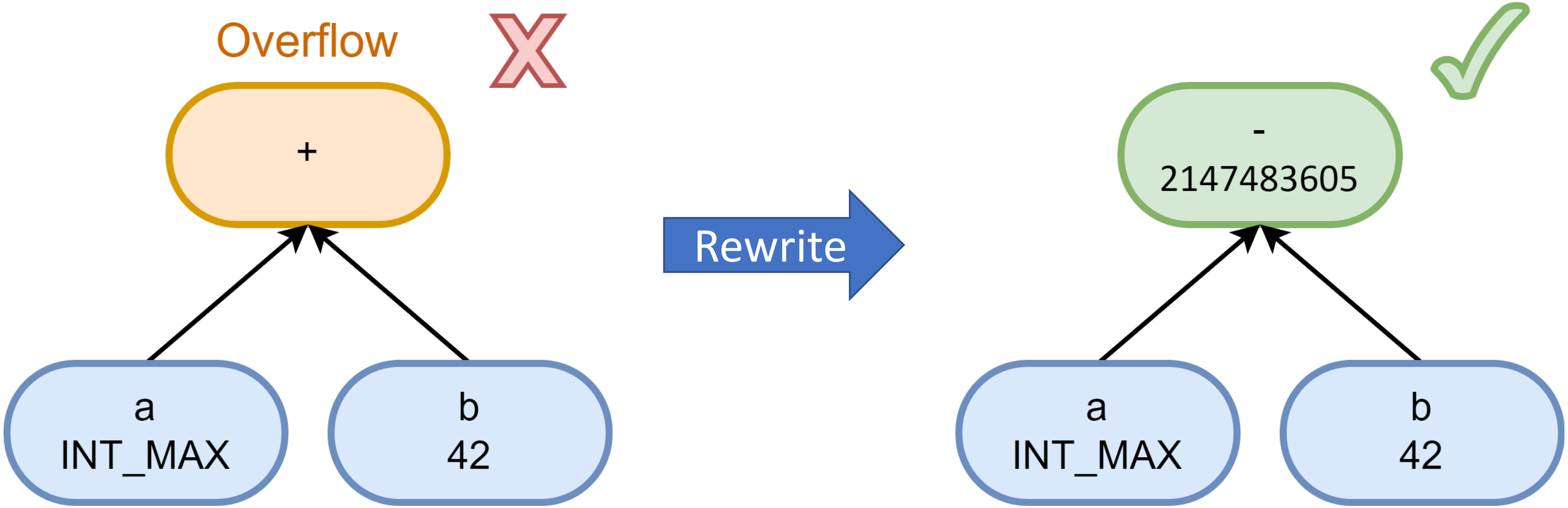
No one!

Program contains UB

https://godbolt.org/z/dfETYae9T

Static Undefined Behavior Avoidance

Based on concrete value tracking and rewrite rules



"Random testing for C and C++ compilers with YARPGen." contains more details

UB Avoidance for Loops

```
var_37 = 20;
var_43 = 99;
…
var_10 = (var_37 / 15) - var_43;
```

```
arr_37[20] = {20, 20, 20, ...};
var_43 = 99;
…
arr_10[0] = (arr_37[0] / 15) - var_43;
```

driver.cpp

```
arr_37[20] = {20, 20, 20, ...};
var_43 = 99;
```

…

test.cpp

```
for (int i = 0; i < 19; ++i) {
  arr_10[i] = (arr_37[i] / 15) - var_43;
}
```

YARPGen Features

- Detect wrong code bugs
  - Avoid Undefined Behavior statically
- Target optimizations explicitly
- Easily extensible for C-family languages
  - Including compilers for emerging languages
- Easy to use

Generation Policies

- IR elements
  - Loop Nest, Loop Sequence, Stencil, Reduction

- Explicit mechanisms
  - Common Subexpression Buffer, Used Constant Buffer

- Skewed Probability
  - Vectorizable Loops, INT_MAX / INT_MIN

The goal is to generate code that is likely to trigger optimization

Loop Fusion and Loop Sequence

```
for (i=0; i < (d ? e : 10); i++)
      a[i] = c[i] + b[i];

for (j=0; j < (d ? e : 10); j++)
    b[j] = b[j] * c[j];
```

```
for (i=0; i < (d ? e : 10); i++){
    a[i] = c[i] + b[i];
    b[i] = b[i] * c[i];
}
```

- Hard to generate purely at random
- Loop Sequence as first-class IR element for synchronized decisions

Loop Patterns: Stencil

```
for (int i = 1; i < n - 1; ++i)
    out[i] = (in[i - 1] +
              in[i] +
              in[i + 1]) / 3;
```

GVN propagates value to next loop iteration

Stencil as a pattern

- arrays
- dimensions
- stride

```
.LBB0_2:
    fadd    d1, d0, d1
    fmov    d2, d0
    ldr     d0, [x9], #8
    fmov    d3, x10
    subs    x8, x8, #1
    fadd    d1, d1, d0
    fmul    d3, d1, d3
    fmov    d1, d2
    str     d3, [x1], #8
    b.ne    .LBB0_2
```

YARPGen Features

- Detect wrong code bugs
    - Avoid Undefined Behavior statically
- Target optimizations explicitly
- Easily extensible for C-family languages
    - Including compilers for emerging languages
- Easy to use

Multi-language Support and IR Lowering

Matrix multiplication

$$c_{ij} = \sum_{k=1}^{K} a_{ik} b_{kj} \; ; i = 1, \ldots, M \; ; j = 1, \ldots, N$$

Multi-language Support and IR Lowering

C++

```
for (int i = 0; i < M; i++)
  for (int j = 0; j < K; j++)
    for (int k = 0; k < N; k++)
      c[i][j] += a[i][k] * b[k][j];
```

ISPC

```
foreach (m = 0 ... M) {
  for (k = 0; k < K; k++) {
    sum = 0.0f;
      for (n = 0; n < N; n++) {
        aValue = a[m*N + n];
        bValue = b[n*K + k];
        sum +=  aValue * bValue;
      }
      c[m*K + k] = sum;
  }
}
```

Multi-language Support and IR Lowering

```
Loop #1: i in [0, 10), step 2
  If-then (d):
    a[i] = b[i] ^ d
  Else:
    a[i] = b[i] & d
Loop #2: j in [0, 10), step 2
  c[i] = b[j] + 134
```

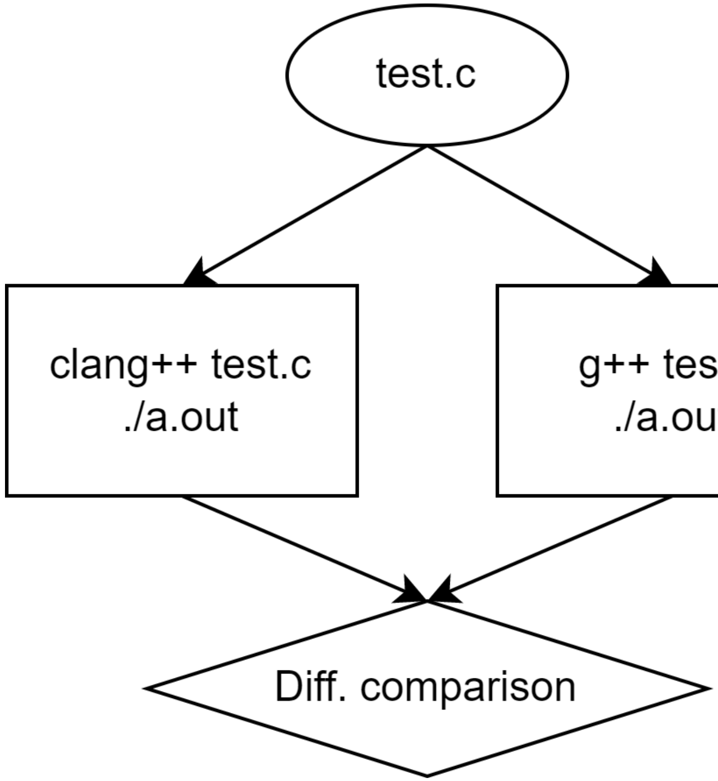
Lowering

```
for (int i = 0; i < 10; i += 2){
  if (d)
    a[i] = b[i] ^ d;
  else
    a[i] = b[i] & d;
}
for (int j = 0; j < 10; j += 2)
  c[i] = b[j] + 134;
```

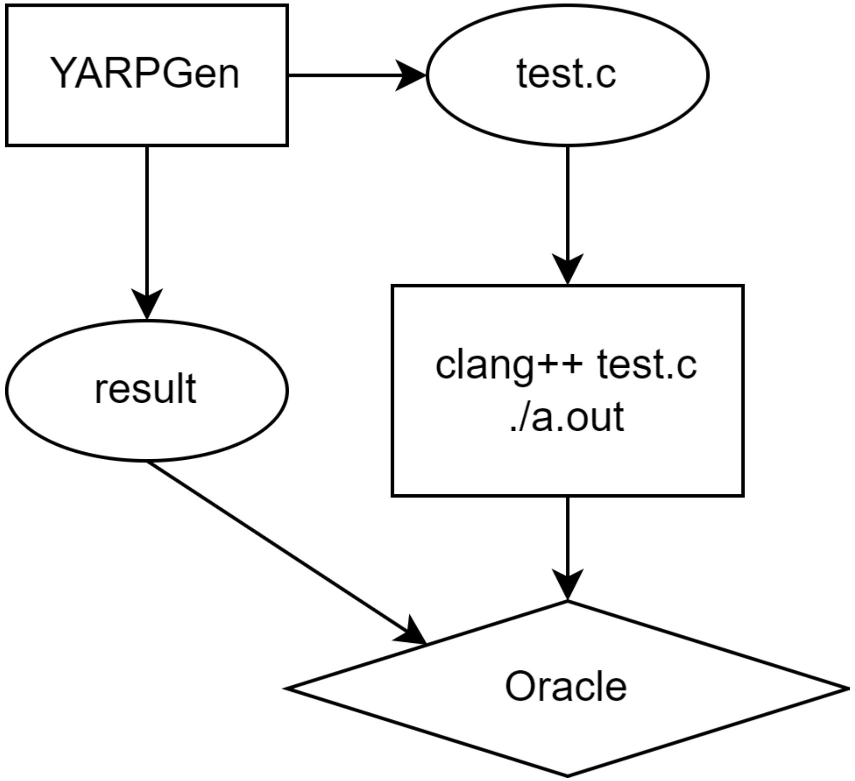
- C-family languages has similar UB rules
- High-level IR is (mostly) independent from target languages
  - contains common information

Test Oracles

Differential testing



Ground truth



YARPGen Features

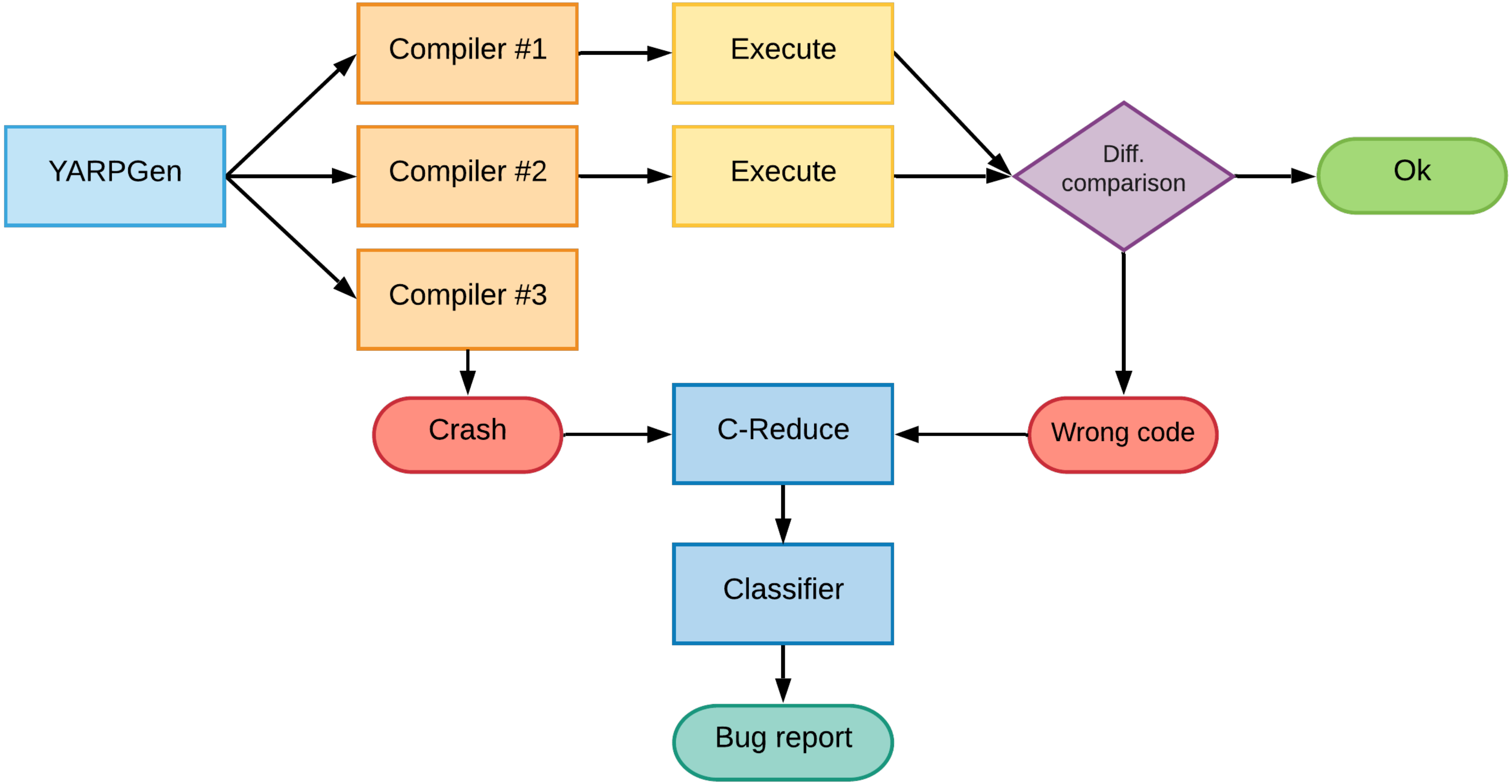
- Detect wrong code bugs
  - Avoid Undefined Behavior statically

- Target optimizations explicitly

- Easily extensible for C-family languages
  - Including compilers for emerging languages

- Easy to use

Automated Testing System

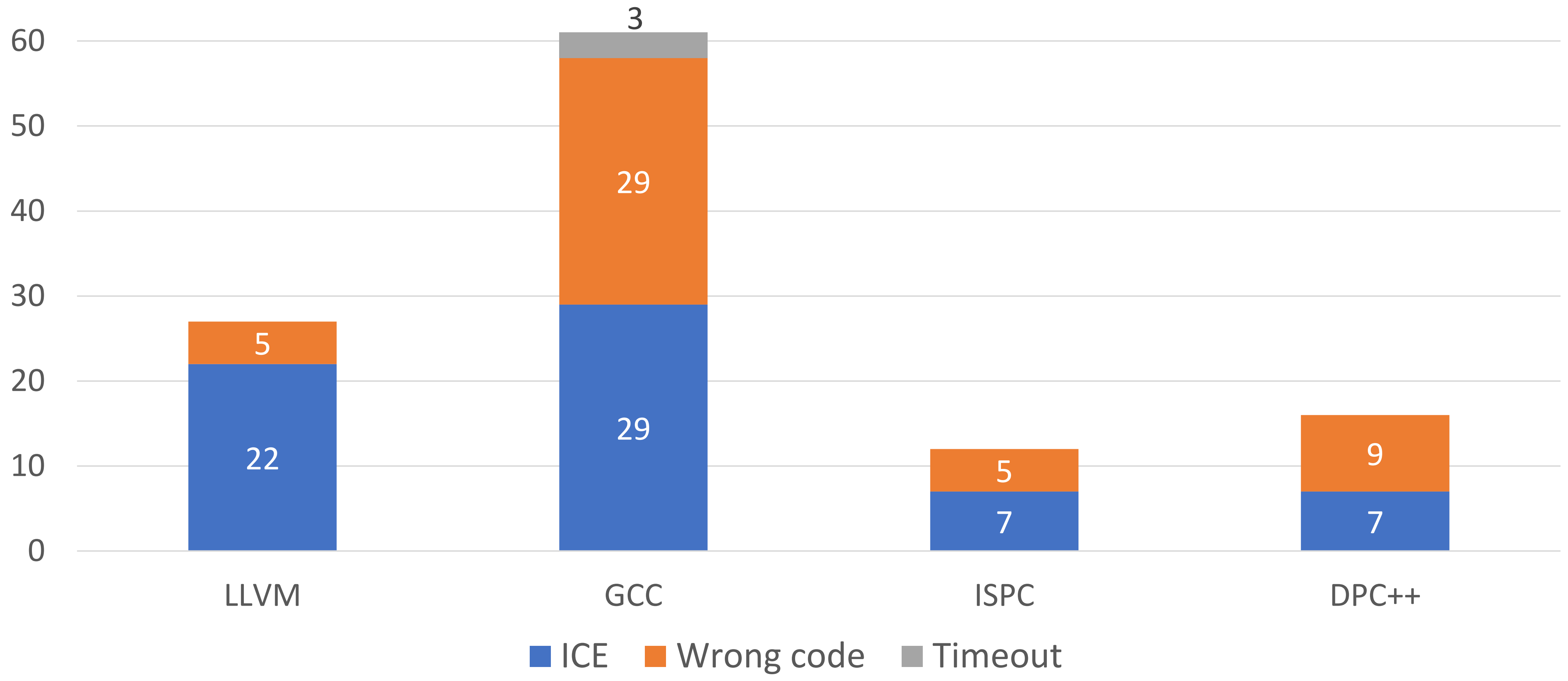


Limitations

- No floating-point support
- Only stdlib function calls
- Lack of dynamic memory allocation
- …

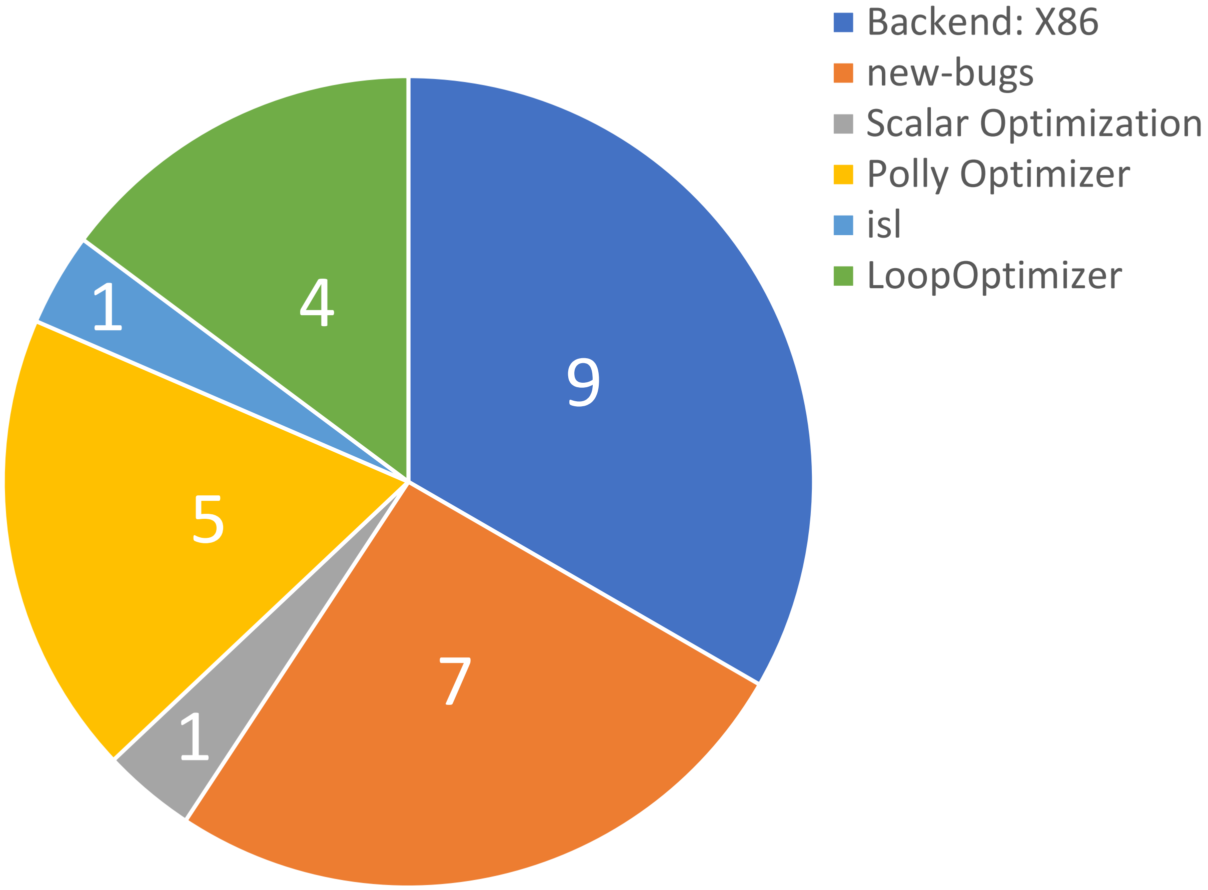
Some are research question; others require more engineering resources

Bugs Distribution by Kind

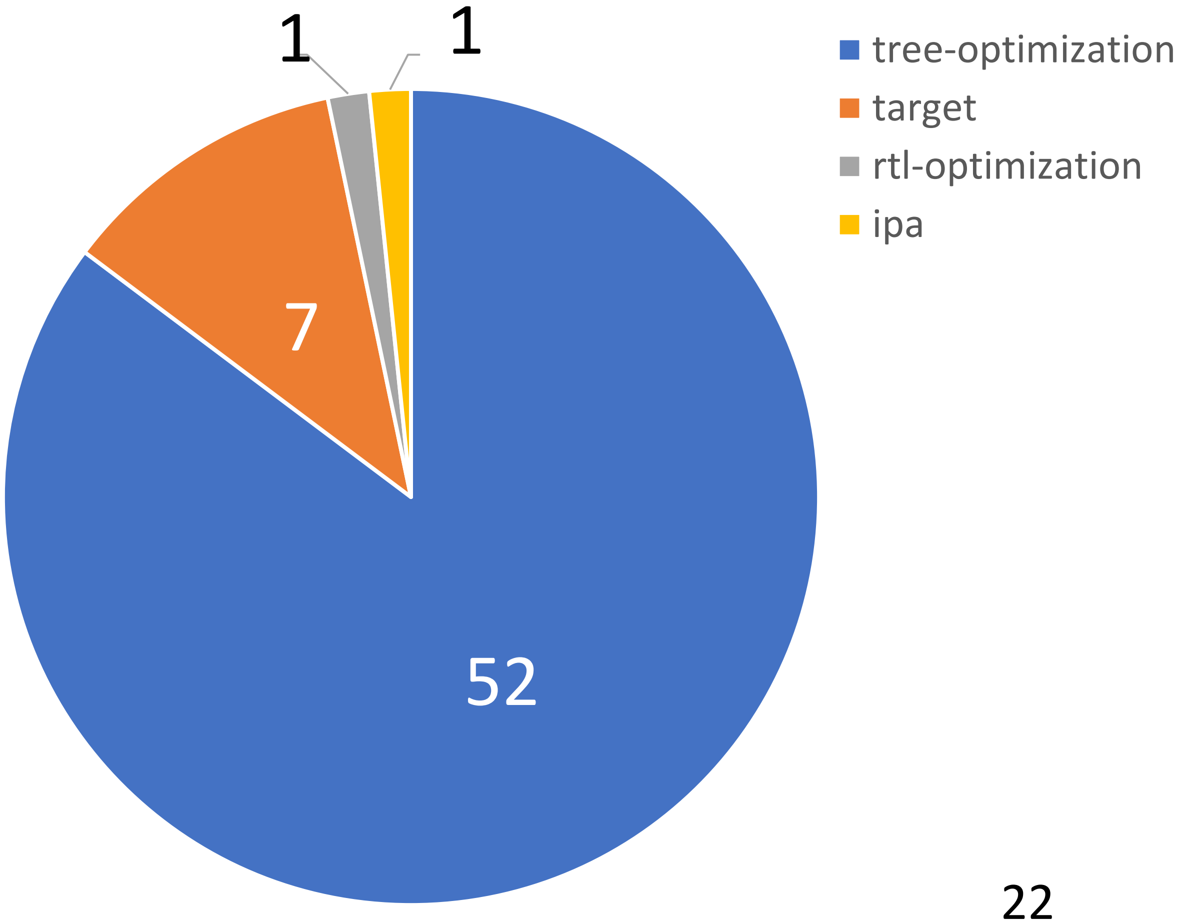


Bugs Distribution by Components

LLVM (27 bugs)



GCC (61 bugs)



Fixed Bugs

- LLVM
  - 70% fixed
    - 18 fixed, 7 new, 1 resolved, 1 confirmed

- GCC
  - 95% fixed
    - 58 fixed, 3 assigned

Test Example

```
/* LoopNest 2 */
for (short i_2 = ((((int) ((short) var_6))) - (181))/*0*/; i_2 < ((((int)
((short) ((((bool) (signed char)4)) && (((bool) ((((((bool) var_2)) ||
(((bool) 3431126726U)))) ? (((unsigned int) ((int) std::max((((unsigned
short) (signed char)-39)), ((unsigned short)63238))))) : (((((bool) arr_2
[i_0] [i_0])) ? (((unsigned int) ((int) (unsigned short)2297))) :
(var_1)))))))))) + (13))/*14*/; i_2 += ((((int) ((short) var_9))) +
(20186))/*3*/) {
  #pragma clang loop vectorize(enable)
  for (long long int i_3 = 0LL/*0*/; i_3 < ((((long long int) var_7)) -
(3048972888LL))/*18*/; i_3 += 2LL/*2*/) {
        arr_15[i_3] = ((int) ((((((unsigned long long int) ((3243476438U) <<
(((int) arr_5 [i_0 / 5]))))) & (((((bool) var_2)) ? (var_8) : (((unsigned
long long int) ((int) arr_12 [i_0] [i_1] [i_2] [i_1] [i_1] [i_1]))))))) <<
(((((int) arr_10 [i_0] [i_1 + 1] [i_2])) << (((int) arr_5 [i_2 / 14]))))));
        arr_16[i_2][i_1] = ((unsigned short) ((unsigned char) ((((int)
arr_10 [i_3] [i_1] [i_2])) & (((int) arr_12 [i_2] [i_1] [i_1 - 3] [i_2]
[i_2] [i_3])))))));
```

LLVM Bug #51677

```
void test() {
#pragma clang loop vectorize_predicate(enable)
  for (char a = 4; a < var_3; a++) {
    arr_13[a] = arr_12[a - 3];
    var_23 = arr_12[a - 1];
  }
}
```

```
>$ clang++ -O0 -march=skx func.cpp driver.cpp && sde -skx -- ./a.out
1
>$ clang++ -O1 -march=skx func.cpp driver.cpp && sde -skx -- ./a.out
0
```

https://github.com/intel/yarpgen

Paper in submission, available upon request

Special thanks to Intel and
LLVM developers,
who fix reported bugs!

Looking for Job

- Expected graduation: end of Spring 2023

- CV: livinskii.com/#cv

- Email: Vsevolod.Livinskii@gmail.com

https://github.com/intel/yarpgen

Paper in submission, available upon request