Debugging Optimized Programs using Black Box Equivalence Checker

Advisor : Prof. Sorav Bansal

- Pratik Karia (2019MCS2568)
- Vaibhav Kiran Kurhe (2019MCS2572)

Problem

- Seamless source-level debugging not possible
- Values optimized out
- Unnatural breakpoint positions
- Issues being present only in optimized version

```c
#include "globals.h"

int s000()
{

//  linear dependence testing
//  no dependence - vectorizable

    for (int i = 0; i < lll; i++) {
      X[i] = Y[i] + val;
    }
  return 0;
}
```

Problem

Unoptimized code

```
88                        for (int i = 0; i < lll; i++) {
(gdb) p i
$1 = 0
(gdb) s
89                              X[i] = Y[i] + val;
(gdb) p i
$2 = 0
(gdb) s
88                        for (int i = 0; i < lll; i++) {
(gdb) p i
$3 = 0
(gdb) s
89                              X[i] = Y[i] + val;
(gdb) p i
$4 = 1
(gdb) s
88                        for (int i = 0; i < lll; i++) {
(gdb) p i
$5 = 1
(gdb)
```

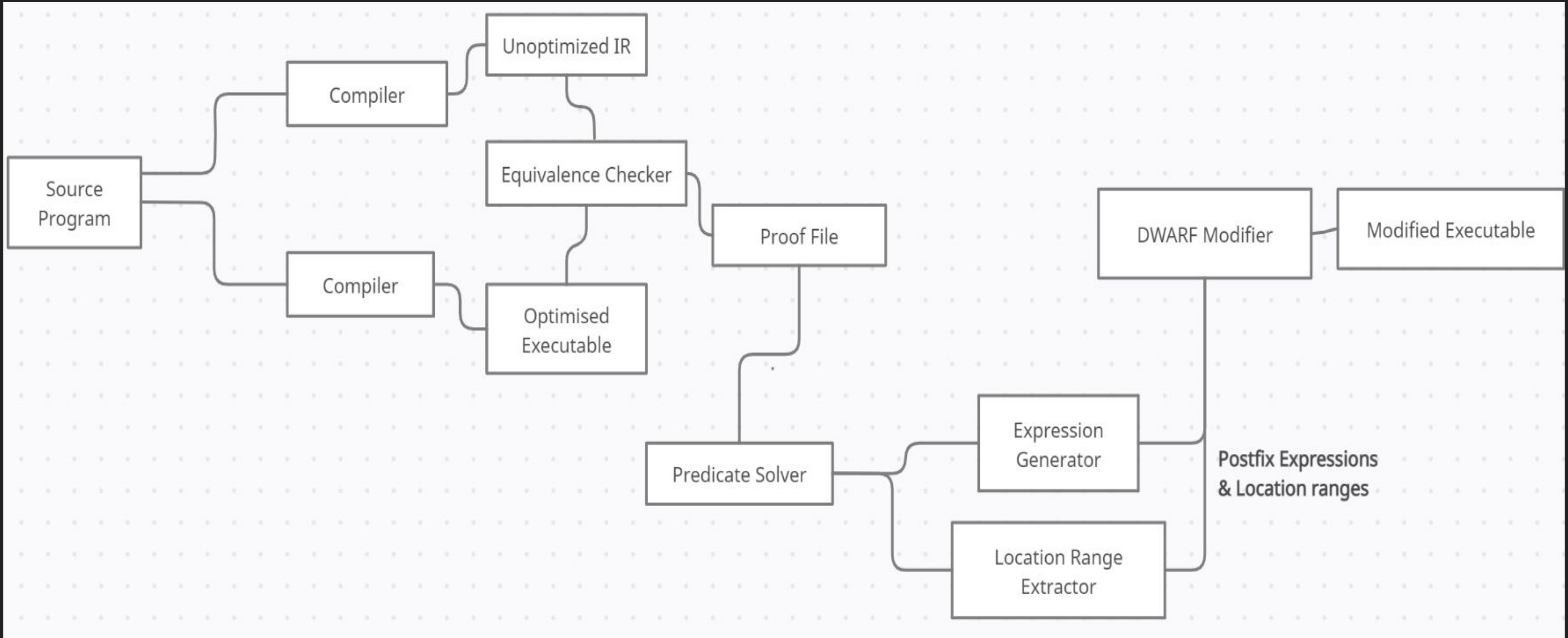
Optimized code

```
88                        for (int i = 0; i < lll; i++) {
(gdb) p i
$1 = 0
(gdb) s
89                              X[i] = Y[i] + val;
(gdb) p i
$2 = 0
(gdb) s
88                        for (int i = 0; i < lll; i++) {
(gdb) p i
$3 = 0
(gdb) s
89                              X[i] = Y[i] + val;
(gdb) p i
$4 = 0
(gdb) s
88                        for (int i = 0; i < lll; i++) {
(gdb) p i
$5 = 0
(gdb) s
89                              X[i] = Y[i] + val;
(gdb) p i
$6 = 0
(gdb)
```

Overview

- Use of equivalence checker to incorporate relevant debugging information
- The predicates in proof file provide a mapping from source (LLVM IR) to destination (optimized executable) variables
- The set of predicates are transformed into a matrix
- Numerical methods are performed on the matrix to get the value for “optimized out” source variables in terms of destination variables
- The resulting values are converted to DWARF expressions and populated into the optimized executables using Gimli (a Rust library)

Tool Architecture



Equivalence Checker

- Produces a mathematical proof of equivalence between an unoptimized and optimized versions of the same source program

- The proof of equivalence consists of predicates of the following format:

```
=pc Lfor.body%1%1_L3%1%0 smallest_point_cover 1 type bv pred 20
=Comment
linear2-32-free_var_idx.23
=LocalSprelAssumptions:
=LhsExpr
1 : input.src.llvm-%i.0 : BV:32
2 : 4 : BV:32
3 : bvmul(1, 2) : BV:32
=RhsExpr
1 : input.dst.exreg.0.0 : BV:32
=predicate done
```

- These predicates are used to generate expressions for source variables

Equivalence Checker

- Other files needed - TFG, ETFG, Harvest file, LLVM-to-source map
- TFG : A Control Flow Graph with Transfer Functions on each edge
- ETFG : Extended TFG suitable for LLVM source program
- Harvest file: Contains mapping of PC labels used in proof file to actual PCs
- LLVM-to-source map : Contains map of LLVM variables to source variables

Predicate Solver

- It mainly consists of 2 modules:
  - Expression Generator - It finds postfix expressions for source variables
  - Location Range Extractor - Finding location range for each predicate

Expression Generator

- The proof file is read and the predicates are extracted from it
- Each predicate is split into arithmetic atoms
- We push all the constants to RHS and the variables to LHS
- Using all the predicates, we get a system of linear equations
- The equations are represented in the form of "AX = B"

Expression Generator (contd.)

- We need values for source variables in terms of destination variables for each PC
- We rearrange X array to bring all source variables before destination variables
- Correspondingly, we rearrange columns of the coefficient matrix (A)
- We create the “Augmented Matrix” and find the “Row Reduced Echelon Form”
- We start from last row and find the values of all destination variables
- We take each source variable and express it in terms of destination variable for each PC

Expression Generator for s000.c

- Example Row reduced echelon form and variable vector

```
1 0 0 5 2
0 1 3 0 3
0 0 1 1 4
0 0 0 1 2
```

```
input.src.llvm-%i.0
input.src.llvm-%j.0
input.dst.exreg.0.1
input.dst.exreg.0.3
```

- Expressions Generated for s000.c for variable i

```
=ZeroAddress
0x0
=TotalPCs
11
=Function
s000_s000
=Expressions
i=1 %eax * 0 + 4 /
```

Location Range Extractor

- Reads the .tfg file to generate a Transfer Function Graph
- For each predicate, extracts the destination registers involved
- Tries to expand the range of PCs over which a predicate is valid
  - Starts from the given PC
  - Gets outgoing edges
  - Expands until we get a back edge or a register modifying instruction

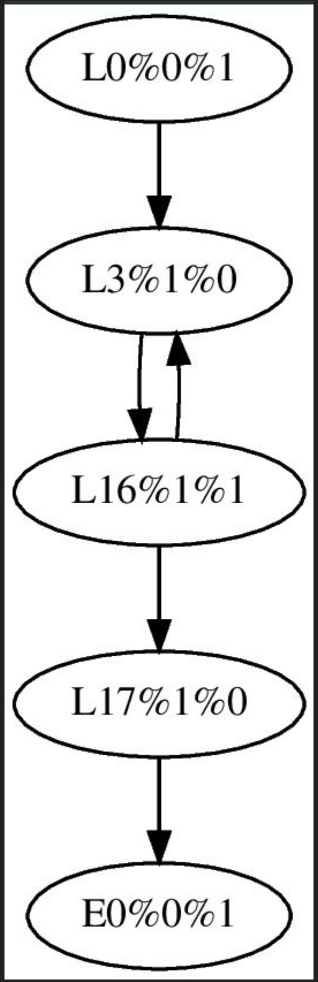
Location Range Extractor Example

```c
#include "globals.h"

int s000()
{

//   linear dependence testing
//   no dependence - vectorizable

    for (int i = 0; i < lll; i++) {
      X[i] = Y[i] + val;
    }
  return 0;
}
```



Valid Predicates Data Flow Analysis

- The need for doing Data Flow Analysis

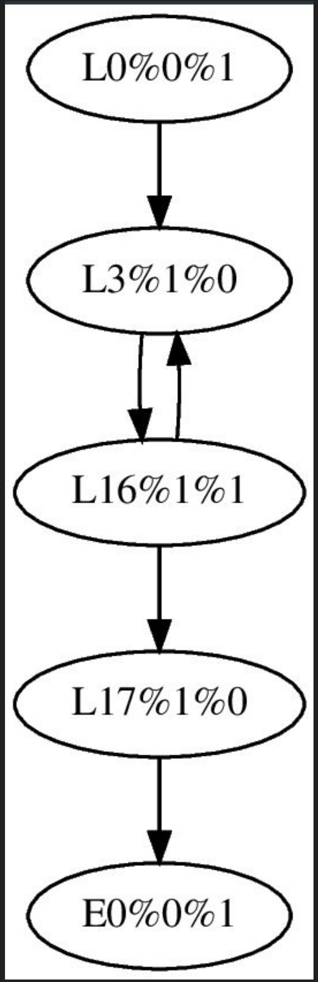
Domain	{ P | where P is a predicate with an LHS and RHS expression and LHS = RHS}
Direction	Forward
Boundary condition	Out[n^{start}] = {}
Initialization to T(top)	In[n] = {} for all non-start nodes

Transfer Function and Meet operator

- Inputs: TFG edge, DFA In value, DFA Out value
- Killing a predicate whose RHS expression contains any modified register
- Using the solutions generated from Expression Generator to simulate GEN set
- Meet operator:
    - Randomly picking any one of the predicates having same source variable

Modifications to DFA

- Consider a simple loop in a program
- A forward edge to loop head with constant predicate
- A backward edge to loop head with non-constant predicate
- Choosing the predicate having non-constant predicate



More precise DFA

- Checking the instruction type in Transfer Function
- Consider a predicate involving the modified register
- Modify the predicate itself if the computation is reversible
    - E.g. Add, Subtract
- Prefer predicates from GEN set over others (backedge and forward edge)
- Adding a Backward DFA pass in sequence

Generated Expression and Location Range

```
=ZeroAddress
0x0
=TotalPCs
11
=Function
s000_s000
=Expressions
i=1 %eax * 0 + 4 /      0x10->0x1b
i=1 %eax 16 - * 0 + 4 /         0x1b->0x2f
```

DWARF Modifier

- Inputs: object file to be modified, new object filename, postfix expressions
- Read and store existing location lists from input object file
- Use DFS to find the DIE for the source variable inside given function
- Construct a DWARF expression from given postfix string expression
- Merge and split if necessary while inserting new DWARF expression
- Add new location list in .debug_loc and point to it in the variable DIE

Object file after debug headers updation

```
 <3><621>: Abbrev Number: 26 (DW_TAG_variable)
    <622>   DW_AT_name        : i
    <624>   DW_AT_decl_file   : 1
    <625>   DW_AT_decl_line   : 9
    <626>   DW_AT_decl_column : 12
    <627>   DW_AT_type        : <0x30>
    <62b>   DW_AT_location    : 0x3e (location list)
 <3><62f>: Abbrev Number: 0
 <2><630>: Abbrev Number: 0
 <1><631>: Abbrev Number: 0
Contents of the .debug_loc section:

    Offset   Begin            End              Expression
    0000003e 00000000 0000000f (DW_OP_lit0; DW_OP_stack_value)
    0000004a 00000010 0000001b (DW_OP_consts: 1; DW_OP_breg0 (eax): 0; DW_OP_mul; DW_OP_consts: 0; DW_OP_plus; DW_OP_consts: 4; DW_OP_div; DW_OP_stack_value)
    00000060 0000001b 0000002d (DW_OP_consts: 1; DW_OP_breg0 (eax): 0; DW_OP_consts: 16; DW_OP_minus; DW_OP_mul; DW_OP_consts: 0; DW_OP_plus; DW_OP_consts: 4; DW_OP_div; DW_OP_stack_value)
    00000079 <End of list>
```

Evaluator Engine

- Inputs: Original optimized object file, Updated object file
- For each object file
  - Traverse through the DIEs in DFS order
  - Classifying DWARF expressions to be constant or non-constant
  - Maintaining the location range and whether it is constant for each variable
- Combining the obtained information from both object files
  - Generate Metrics as shown in the Results Table

Results (TSVC benchmarks, clang/gcc/icc)

Table (1a)

F_n	I_m p/v	M p/v	T	A_v b_f/a_f
s000	14/1	-/-	19	17/17
s1112	22/1	-/-	27	25/25
s1119	8/1	-/-	17	20/20
s112	-/-	10/1	13	0/10
s116	1/1	13/1	17	1/14
s119	12/1	-/-	38	52/52
s121	17/1	-/-	40	60/60
s1221	12/1	-/-	16	12/12
s122	3/1	12/2	17	63/75
s1251	17/1	-/-	20	17/17
s131	17/1	-/-	37	70/70
s132	21/1	-/-	53	210/210
s1351	14/1	15/3	17	14/59
s162	-/-	41/1	53	54/95
s171	18/1	7/1	41	66/73
s173	14/1	-/-	17	30/30
s2244	15/1	-/-	47	44/44
s243	16/1	-/-	51	48/48

clang results

Table (1b)

F_n	I_m p/v	M p/v	T	A_v b_f/a_f
s000	1/1	6/1	11	3/9
s1111	-/-	19/1	20	0/19
s1112	1/1	7/1	12	3/10
s111	-/-	28/1	28	0/28
s112	-/-	22/1	22	0/22
s113	-/-	7/1	20	9/16
s119	1/1	26/2	27	12/44
s121	-/-	18/1	18	11/29
s1221	-/-	5/1	9	1/6
s122	6/1	14/2	17	55/76
s1251	-/-	12/1	13	0/12
s125	-/-	20/3	24	6/49
s127	-/-	16/2	17	1/32
s1281	-/-	16/1	17	0/16
s128	1/1	19/2	20	2/38
s131	-/-	15/1	15	15/30
s132	-/-	28/1	28	84/112
s1351	-/-	8/4	9	0/29
s162	1/1	8/1	43	62/70
s173	-/-	8/1	9	9/17
s174	-/-	48/1	64	64/112
s2233	-/-	37/1	39	13/48
s2244	-/-	24/1	24	0/24
s243	-/-	19/1	21	0/19

gcc results

I_m : Improved
M : Missing
p : PCs
v : Count of variables
T : Total PCs
A_v : Available cumulative debug info
b_f : Before updation
a_f : After updation

Table (2a)

F_n	I_m p/v	M p/v	T	A_v b_f/a_f
s114	-/-	46/2	47	0/79
s124	-/-	44/1	50	50/94
s125	-/-	36/1	37	74/110
s127	-/-	57/1	63	63/120
s252	-/-	9/1	19	19/28

icc results

Results (Larger Example - 1)

```c
#include "globals.h"

extern void dummy_function();

int example1(int arg1)
{
  int l = 10 * arg1;
  for (int i = 0; i < lll; i++) {
    X[i] = Y[i] + val;
  }
  dummy_function();
  for (int j = 0; j < lll; j++) {
    Y[j] = X[j] - val + l;
  }
  dummy_function();
  for (int k = 0; k < lll; k++) {
    X[k] = X[k] + val - l;
    ++l;
  }
  return 0;
}
```

```
=ZeroAddress
0x0
=TotalPCs
49
=Function
example1
=Expressions
arg1=1 -1 %ebx * 0 + -10 / * 0 +  0x20->0x58
arg1=1 -1 %ebx * 0 + -10 / * 0 +  0x6e->0xb0
arg1=1 bvextract(%xmm6, 31, 0) * 0 + 10 /   0xb0->0xda
i=4 -1 %eax * 0 + -16 / * 0 +   0x1a->0x2b
i=4 -1 %eax 16 - * 0 + -16 / * 0 +  0x2b->0x42
j=4 -1 %eax * 0 + -16 / * 0 +   0x50->0x63
j=4 -1 %eax 16 - * 0 + -16 / * 0 +  0x63->0x7a
k=4 -1 %eax * -905315344 + -16 / * 0 +  0x83->0xbf
k=4 -1 %eax 16 - * -905315344 + -16 / * 0 +   0xbf->0xd9
l=1 bvextract(%xmm6, 31, 0) * 4 -1 %eax * -905315344 + -16 / * + 0 +  0x83->0xbf
l=1 bvextract(%xmm6, 31, 0) * 4 -1 %eax 16 - * -905315344 + -16 / * + 0 +   0xbf->0xd9
```

F_n	I_m(p/v)	M(p/v)	T	$A_v(b_f/a_f)$
Example1	7/2	27/3	52	94/121

Results (Larger Example - 2)

```c
#include "globals.h"

extern void dummy_function();

int example2(int arg1, int arg2)
{
  int l = 10 * arg1;
  int m = 20 * arg2;
  for (int i = 0; i < LEN; i++) {
    a[i] = a[i] * b[i] * c[i] + val;
  }
  dummy_function();
  for (int j = 0; j < lll; j++) {
    c[j] = b[j] - m;
    m += 2;
  }
  dummy_function();
  for (int k = 0; k < lll; k++) {
    b[k] = b[k] + val - l;
    --l;
  }
  return 0;
}
```

```
=ZeroAddress
0x0
=TotalPCs
58
=Function
example2
=Expressions
arg1=1 %ebx * 0 + 10 /  0x24->0x64
arg1=1 %ebx * 0 + 10 /  0x80->0xaa
arg1=1 -1 %ebx * 0 + -10 / * 0 +  0x5f->0x80
arg1=1 -1 %ebx * 0 + -10 / * 0 +  0xa5->0xe0
arg1=1 -1 bvextract(%xmm1, 31, 0) * 0 + -10 / * 0 +   0xe0->0x104
arg2=1 %esi * 0 + 20 /  0x30->0x64
arg2=1 -1 %esi * 0 + -20 / * 0 +  0x5f->0x80
arg2=1 -1 %esi * 0 + -20 / * 0 +  0x93->0x10b
i=4 -1 %eax * 0 + -16 / * 0 +   0x22->0x3b
i=4 -1 %eax 16 - * 0 + -16 / * 0 +  0x3b->0x64
k=0   0x0->0x10b
l=10 -1 bvextract(%xmm1, 31, 0) * 0 + -10 / * 0 +   0xcd->0x104
m=1 bvextract(%xmm1, 31, 0) * 0 +   0x80->0xa5
```

F_n	I_m(p/v)	M(p/v)	T	$A_v(b_f/a_f)$
Example2	-/-	24/2	62	188/212

Questions/Suggestions?

Thank You