POLITECNICO MILANO 1863
NECST laboratory

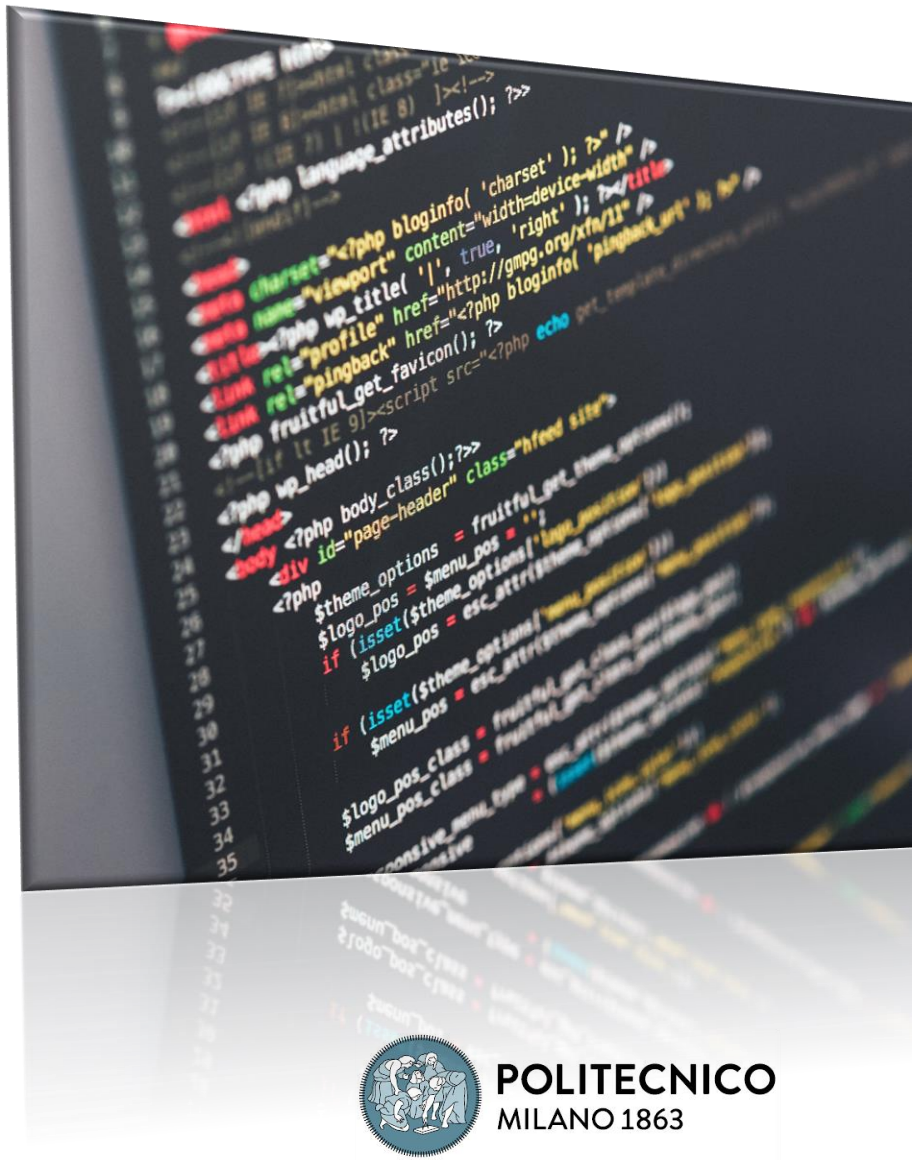
POLITECNICO
MILANO 1863

Integer Range Analysis

Maglione Sandro
June 29th, 2020

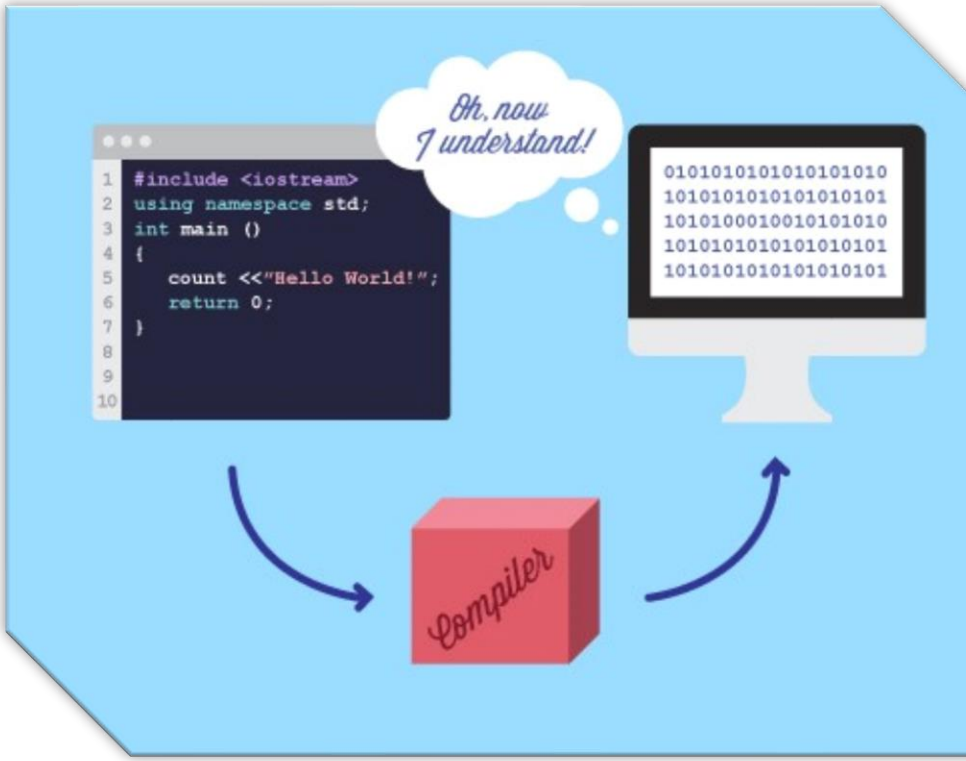
Context

Source codes are made of millions of lines of code, problem of memory and resources!



Context

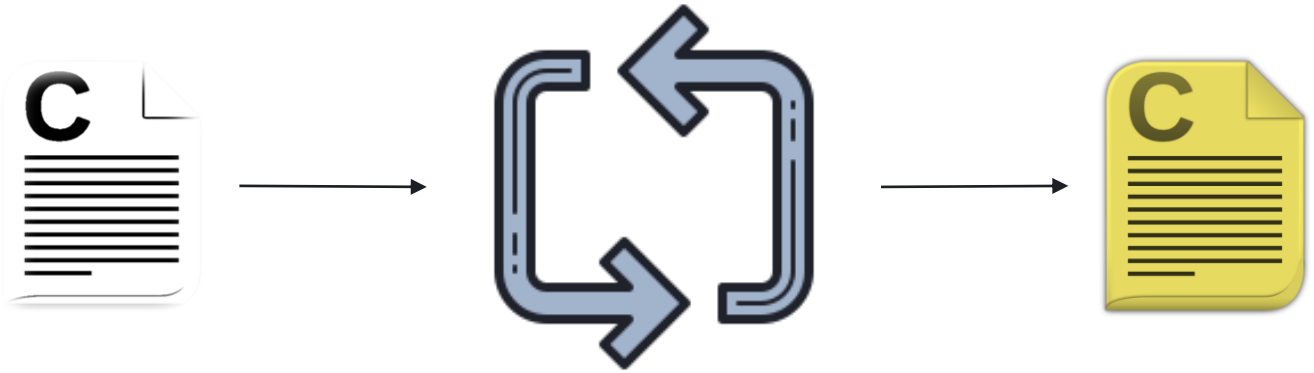
Optimization of resources using compiler and static analysis



Context

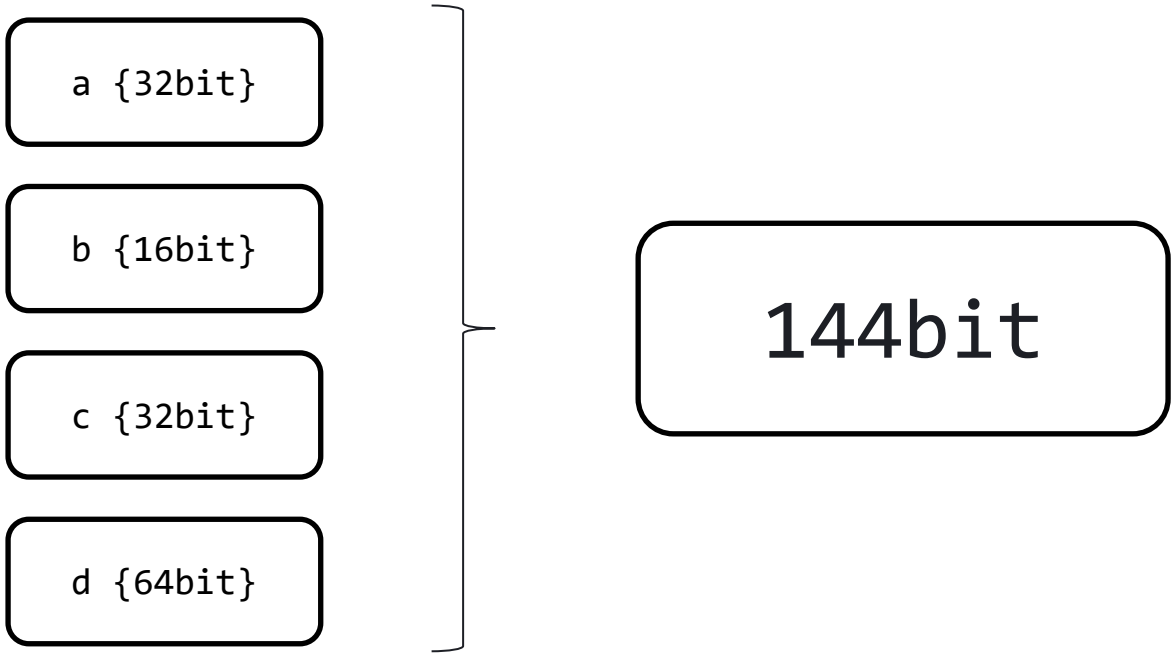
Bitwidth analysis, reduce number of bits allocated for each variable

How to optimize?

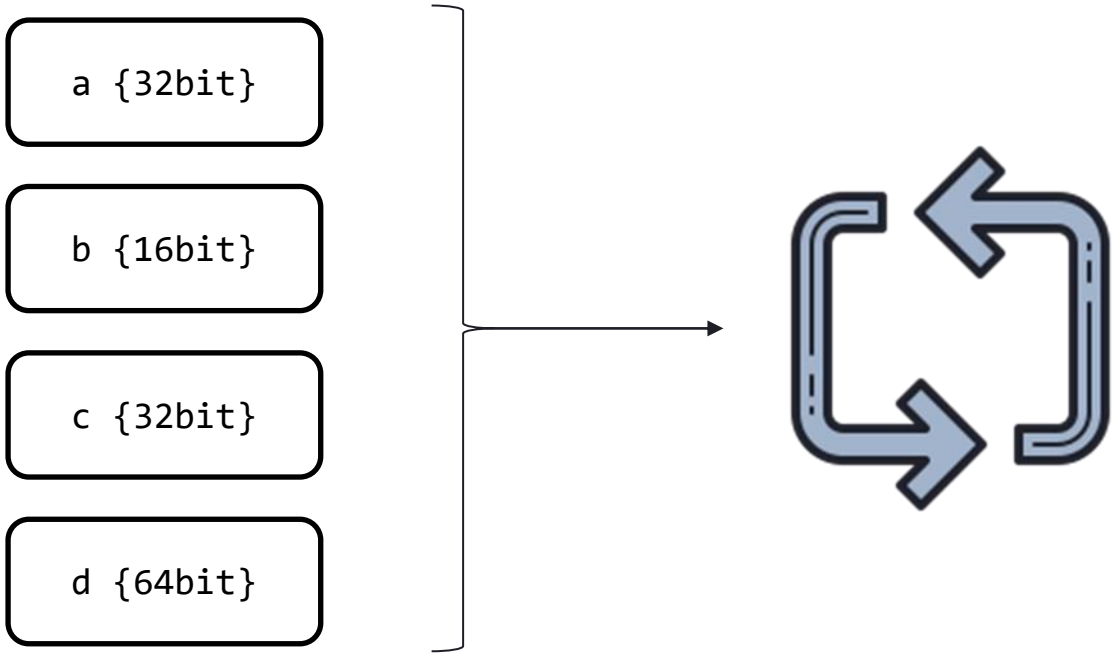


Static analysis, reduce memory usage while keeping the code correctness

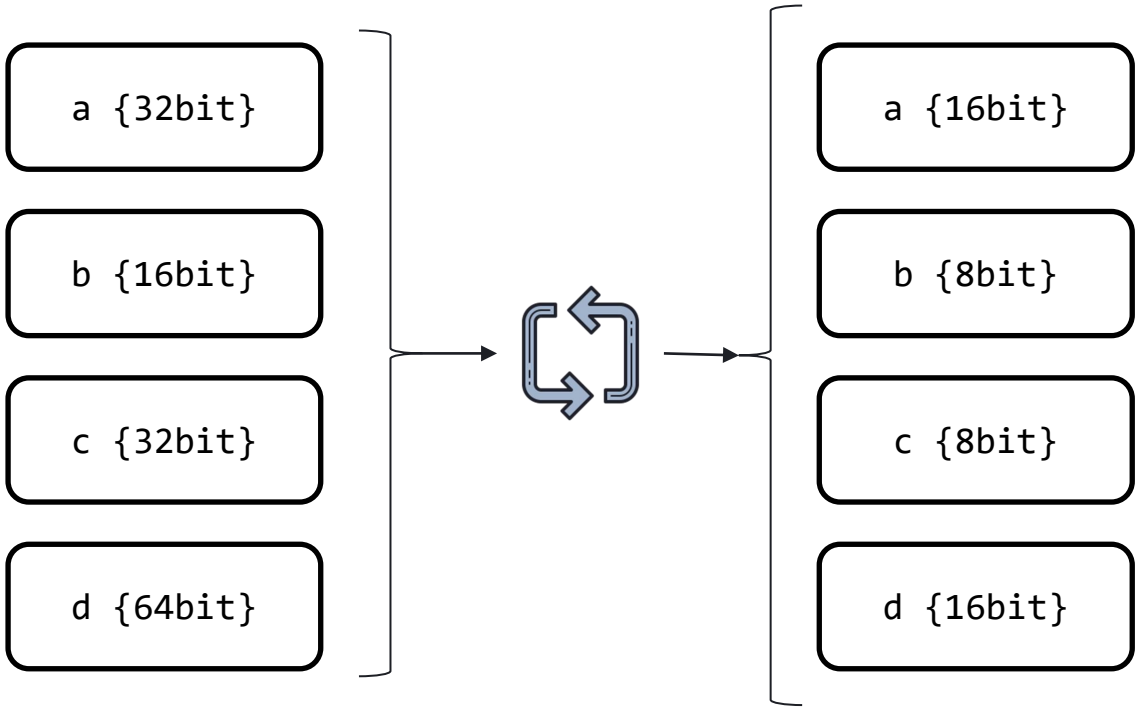
Value Range Analysis



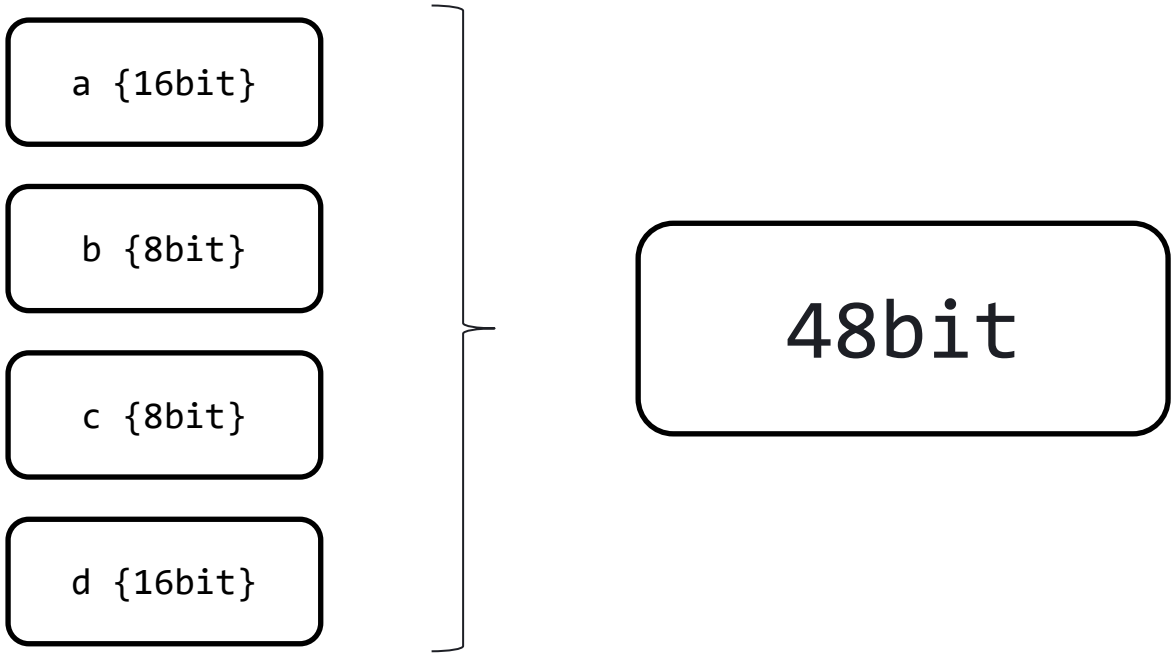
Value Range Analysis



Value Range Analysis



Value Range Analysis



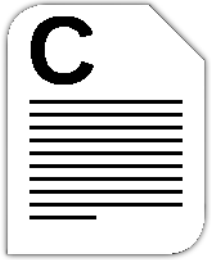
Purpose

- Optimize the allocation of memory for all variables in the code
- Infer the minimum and maximum values that each variable can assume during runtime
- Compute number of bits to allocate for each variable based on its maximum range

LLVM

- Open-source compiler infrastructure project
- Intermediate representation (IR)
- LLVM Pass Framework
  - Transformations
  - Optimizations
  - Analysis

LLVM



```
define dso_local i32 @main() #0 {
entry:
  %retval = alloca i32, align 4
  %a = alloca i32, align 4
  %b = alloca i32, align 4
  %c = alloca i32, align 4
  %d = alloca i32, align 4
  store i32 0, i32* %retval, align 4
  store i32 0, i32* %a, align 4
  store i32 10, i32* %b, align 4
  %0 = load i32, i32* %a, align 4
  %add = add nsw i32 %0, 3
  store i32 %add, i32* %c, align 4
  %1 = load i32, i32* %b, align 4
  %add1 = add nsw i32 %1, 10
  store i32 %add1, i32* %d, align 4
  ret i32 0
}
```

LLVM

```
define dso_local i32 @fun() #0 {
entry:
  %j = alloca i32, align 4
  %a = alloca i32, align 4
  store i32 10, i32* %j, align 4
  store i32 0, i32* %a, align 4
  br label %for.cond

for.cond:
    ; preds = %for.inc, %entry
  %0 = load i32, i32* %j, align 4
  %cmp = icmp sgt i32 %0, 0
  br i1 %cmp, label %for.body, label %for.end

for.body:
    ; preds = %for.cond
  %1 = load i32, i32* %a, align 4
  %sub = sub nsw i32 %1, 5
  store i32 %sub, i32* %a, align 4
  br label %for.inc

for.inc:
    ; preds = %for.body
  %2 = load i32, i32* %j, align 4
  %dec = add nsw i32 %2, -1
  store i32 %dec, i32* %j, align 4
  br label %for.cond

for.end:
    ; preds = %for.cond
  %3 = load i32, i32* %a, align 4
  ret i32 %3
}
```

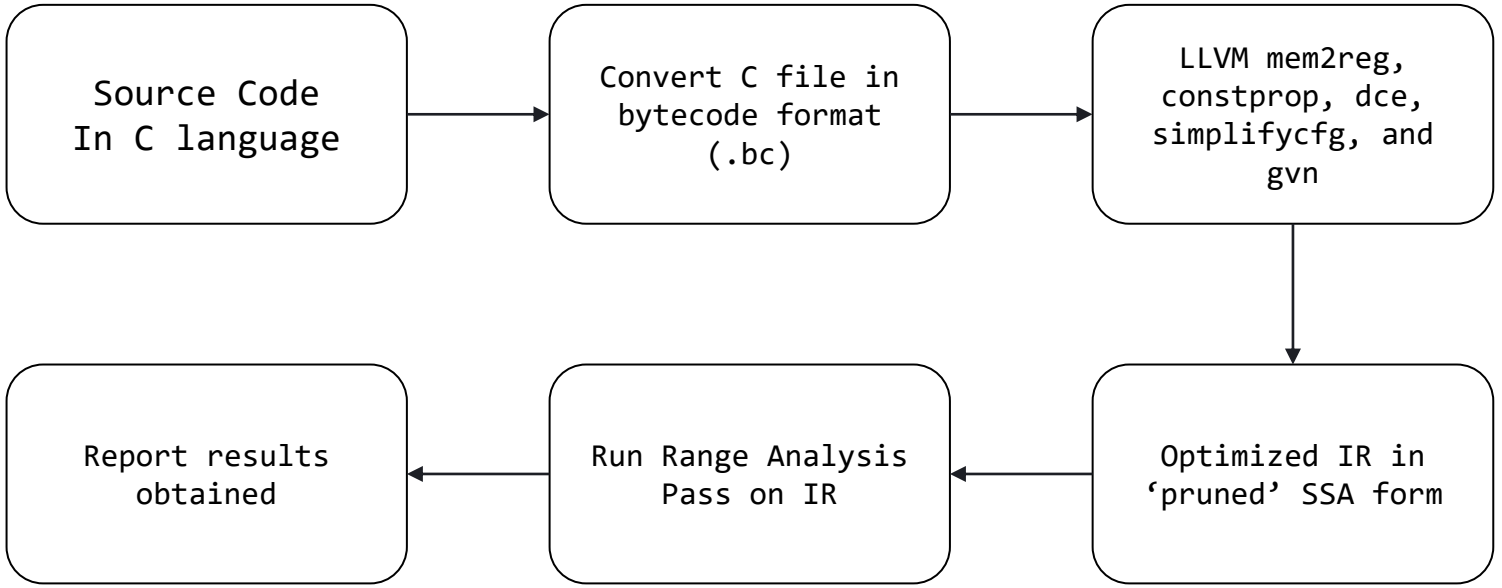


```
define dso_local i32 @fun() #0 {
entry:
  br label %for.cond

for.cond:
    ; preds = %for.inc, %entry
  %a.0 = phi i32 [ 0, %entry ], [ %sub, %for.inc ]
  %j.0 = phi i32 [ 10, %entry ], [ %dec, %for.inc ]
  %cmp = icmp sgt i32 %j.0, 0
  br i1 %cmp, label %for.body, label %for.end

for.body:
    ; preds = %for.cond
  %sub = sub nsw i32 %a.0, 5
  br label %for.inc

for.inc:
    ; preds = %for.body
  %dec = add nsw i32 %j.0, -1
  br label %for.cond

for.end:
    ; preds = %for.cond
  ret i32 %a.0
}
```

LLVM

```
define dso_local i32 @fun() #0 {
entry:
  br label %for.cond

for.cond:
    ; preds = %for.inc, %entry
  %a.0 = phi i32 [ 0, %entry ], [ %sub, %for.inc ]
  %j.0 = phi i32 [ 10, %entry ], [ %dec, %for.inc ]
  %cmp = icmp sgt i32 %j.0, 0
  br i1 %cmp, label %for.body, label %for.end

for.body:
    ; preds = %for.cond
  %sub = sub nsw i32 %a.0, 5
  br label %for.inc

for.inc:
    ; preds = %for.body
  %dec = add nsw i32 %j.0, -1
  br label %for.cond

for.end:
    ; preds = %for.cond
  ret i32 %a.0
}
```

Methodology



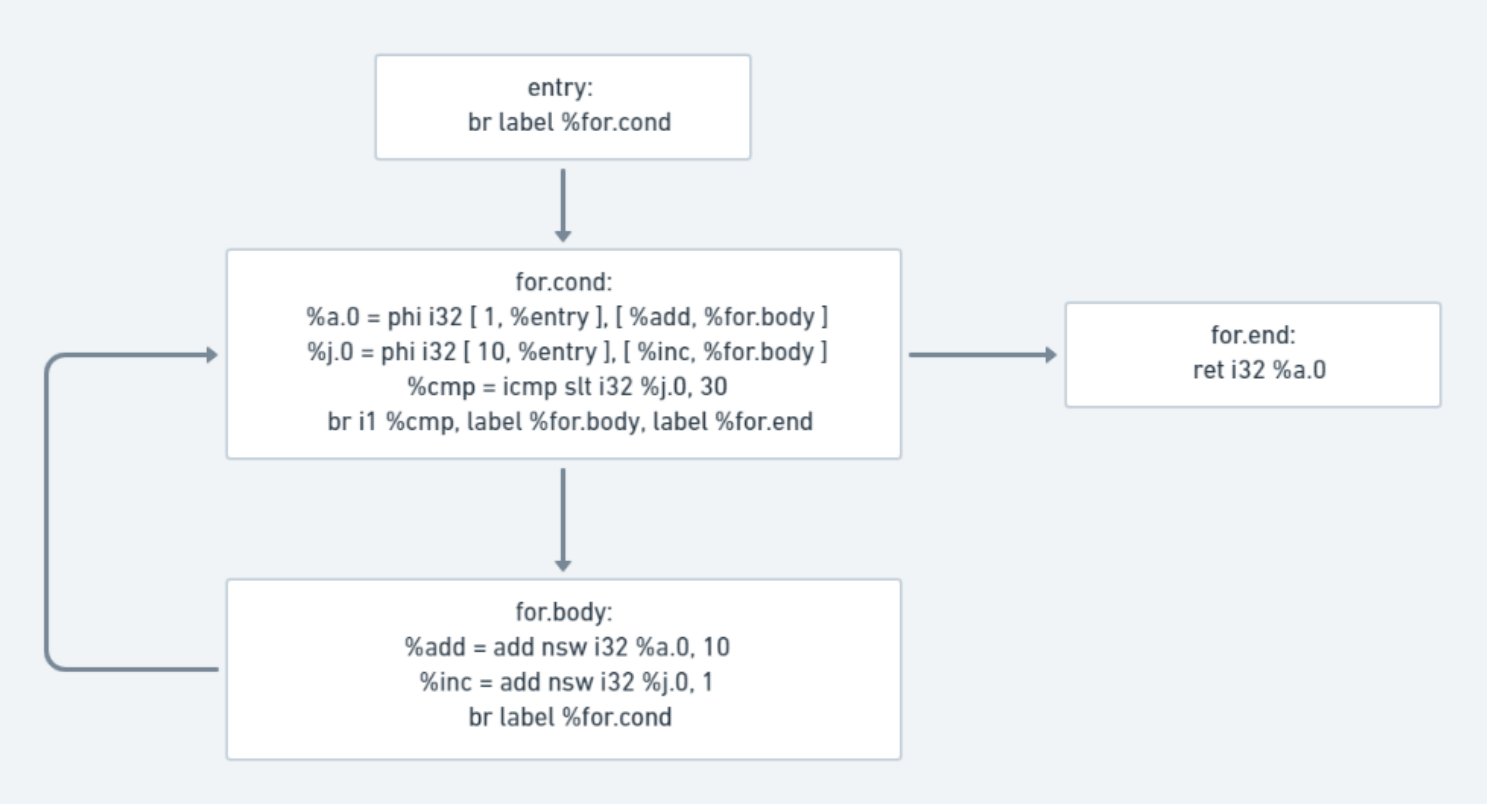
Execution

```
1 int fun()
2 {
3     int j = 10;
4     int a = 1;
5
6     for (; j < 30; ++j)
7     {
8         a = a + 10;
9     }
10
11    return a;
12 }
```



```
define dso_local i32 @fun() #0 {
entry:
  br label %for.cond

for.cond:
    ; preds = %for.body, %entry
  %a.0 = phi i32 [ 1, %entry ], [ %add, %for.body ]
  %j.0 = phi i32 [ 10, %entry ], [ %inc, %for.body ]
  %cmp = icmp slt i32 %j.0, 30
  br i1 %cmp, label %for.body, label %for.end

for.body:
    ; preds = %for.cond
  %add = add nsw i32 %a.0, 10
  %inc = add nsw i32 %j.0, 1
  br label %for.cond

for.end:
    ; preds = %for.cond
  ret i32 %a.0
}
```

Execution


entry:
br label %for.cond
for.cond:
%a.0 = phi i32 [1, %entry], [%add, %for.body]
%j.0 = phi i32 [10, %entry], [%inc, %for.body]
%cmp = icmp slt i32 %j.0, 30
br i1 %cmp, label %for.body, label %for.end
for.end:
ret i32 %a.0
for.body:
%add = add nsw i32 %a.0, 10
%inc = add nsw i32 %j.0, 1
br label %for.cond

Execution

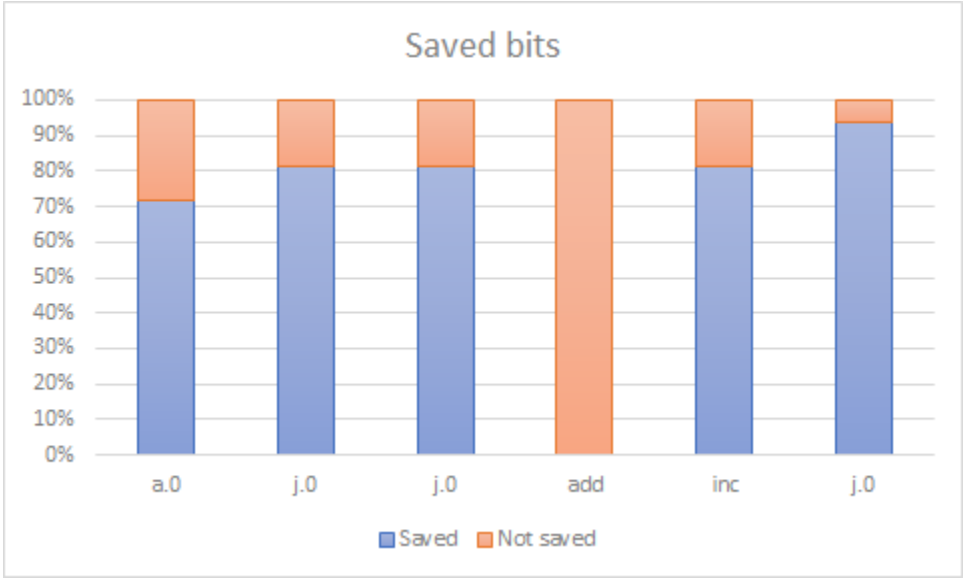
```
1 int fun()
2 {
3     int j = 10;
4     int a = 1;
5
6     for (; j < 30; ++j)
7     {
8         a = a + 10;
9     }
10
11    return a;
12 }
```



```
--- VALUE-RANGES ---
BB: entry

BB: for.cond
    a.0(1, 201) = 201 {9bit}
    j.0(10, 30) = 21 {6bit}

BB: for.body
    j.0(10, 29) = 20 {6bit}
    add(-Inf, +Inf) = MAX
    inc(11, 30) = 20 {6bit}

BB: for.end
    j.0(30, 30) = 1 {2bit}
```

Experimental Results

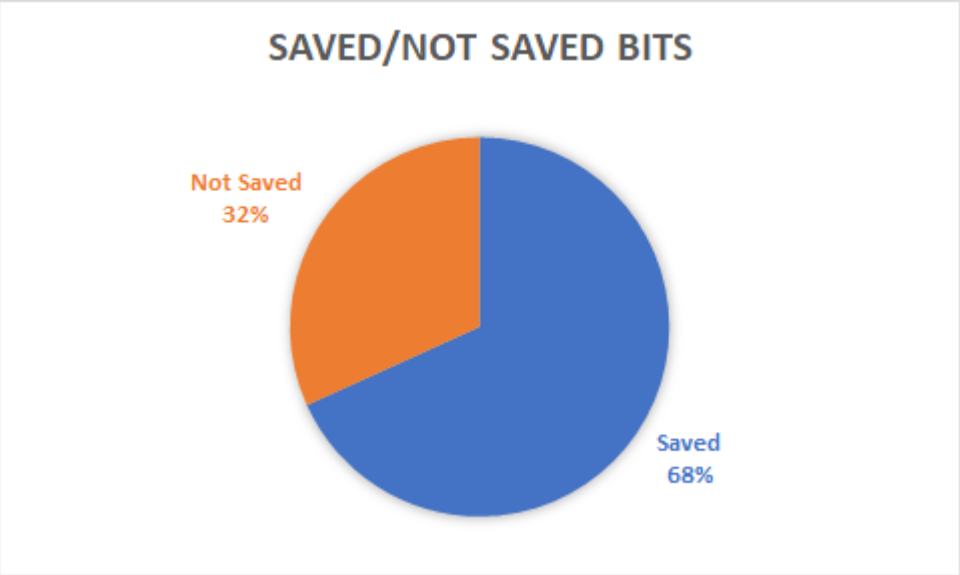
```
—— VALUE-RANGES ——
BB: entry

BB: for.cond
    a.0(1, 201) = 201 {9bit}
    j.0(10, 30) = 21 {6bit}

BB: for.body
    j.0(10, 29) = 20 {6bit}
    add(-Inf, +Inf) = MAX
    inc(11, 30) = 20 {6bit}

BB: for.end
    j.0(30, 30) = 1 {2bit}
```



Performance achieved on the example source code.

Experimental Results

```
—— VALUE-RANGES ——
BB: entry

BB: for.cond
    a.0(1, 201) = 201 {9bit}
    j.0(10, 30) = 21 {6bit}

BB: for.body
    j.0(10, 29) = 20 {6bit}
    add(-Inf, +Inf) = MAX
    inc(11, 30) = 20 {6bit}

BB: for.end
    j.0(30, 30) = 1 {2bit}
```



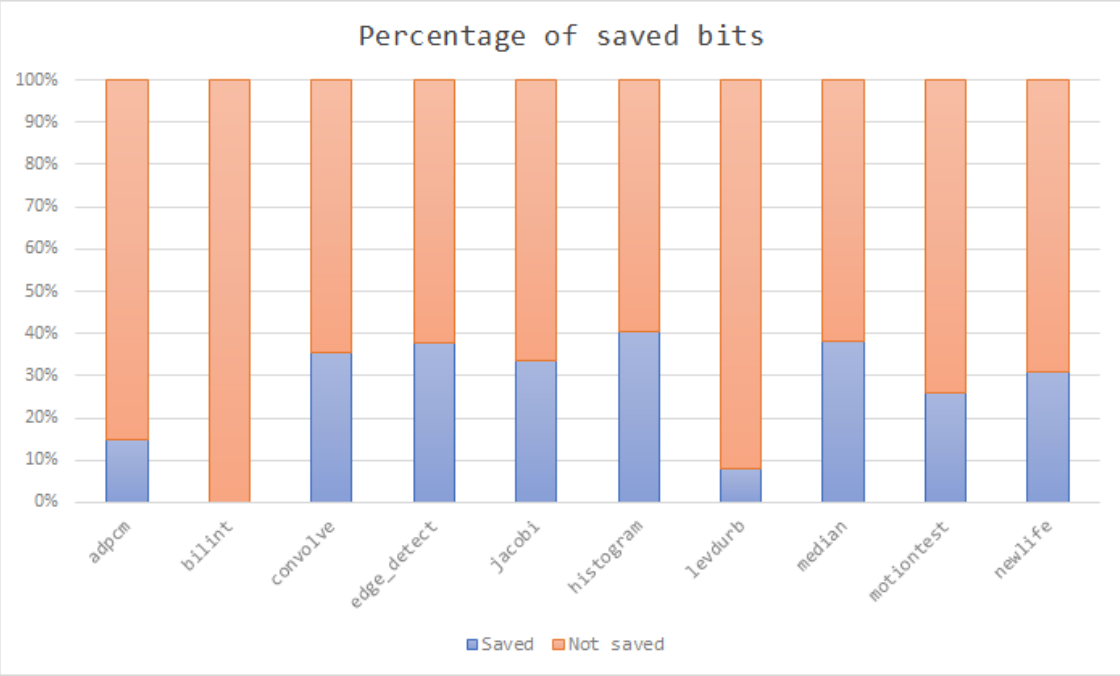
Performance achieved on the example source code.

Experimental Results

Benchmark	Lines	Performance (%)	Saved bits
adpcm	196	14.72	245
bilint	114	0.00	0
bubblesort	69	34.76	356
convolve	81	35.49	727
edge_detect	182	37.63	289
histogram	110	33.75	594
jacobi	68	40.55	571
levdurb	50	8.06	111
median	99	38.12	427
motiontest	44	26.08	217
newlife	121	30.71	747

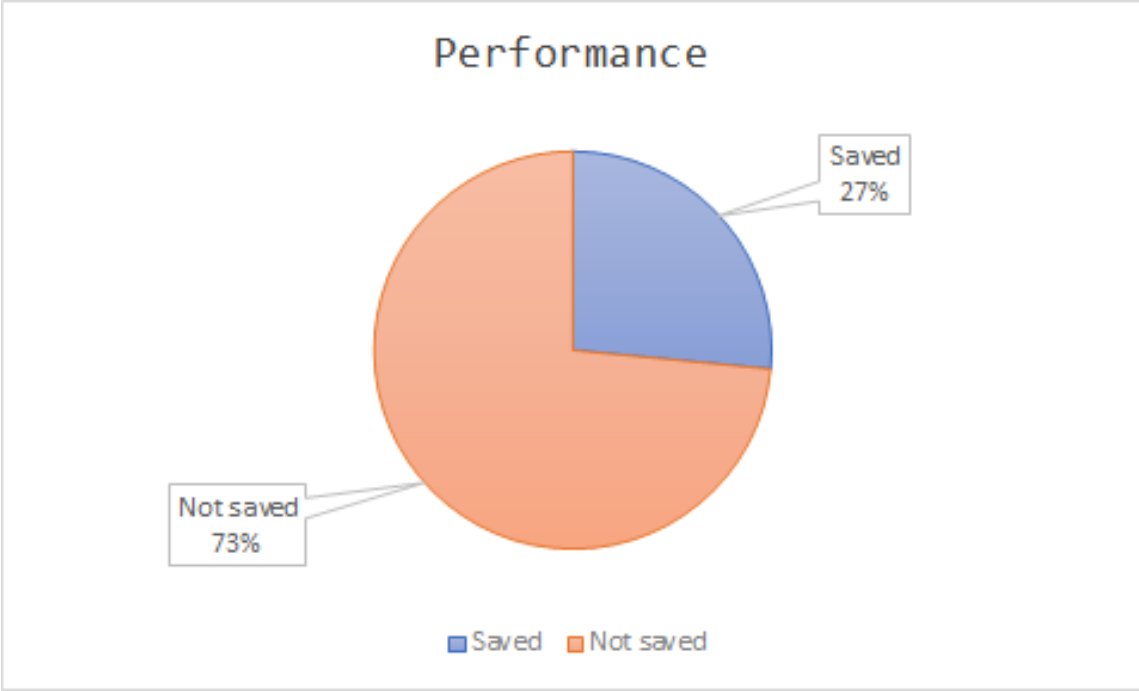
Performance achieved on the bitwise benchmark set.

Experimental Results



Performance achieved on the bitwise benchmark set.

Experimental Results



Performance achieved on the bitwise benchmark set.

Related Works

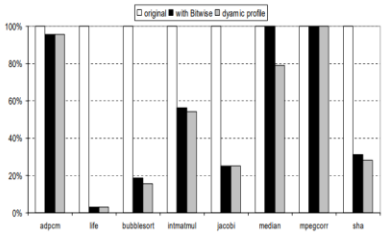
1. A Fast and Low-Overhead Technique to Secure Programs Against Integer Overflows Fernando Magno Quintao Pereira Raphael Ernani Rodrigues Victor Hugo Sperle Campos

   Eliminated 24.93% of the integer overflow checks

2. The Design and Implementation of a Non-Iterative Range Analysis Algorithm on a Production Compiler Douglas do Couto Teixeira and Fernando Magno Quintao Pereira

   Increased 1-bit variables by 1.04x, 8-bit variables by 52.3x and 16-bit variables by 26.6x

3. Bitwidth Analysis with Application to Silicon Compilation Mark Stephenson Jonathan Babb and Saman Amarasinghe



Maglione Sandro

10532096 – 940807

674339166

sandro.maglione@mail.polimi.it