

Instruction Combining

Jovan Škorić 362/2020
Nikola Kuburović 369/2020

UVOD

Instruction combining je tehnika kombinovanja instrukcija radi smanjivanja ukupnog broja, kao i radi formiranja jednostavnijih instrukcija.

Primeri:

1) Optimizacija 1

$y = x + 1;$

$z = y + 1;$

menjamo u

$z = x + 2;$

2) Optimizacija 2

$x++;$

$x++;$

menjamo u

$x += 2;$

3) Optimizacija 3

$\text{int } x = 0;$

$x += 2;$

menjamo u

$\text{int } x = 2;$

Pass pipeline se sastoji od sledećih pass-ova:

- 1) RHS Move
- 2) Convert compares
- 3) Replace compares
- 4) Replace same-operand Add with Shift
- 5) Replace powers of 2 with Shift
- 6) Neka od optimizacija (i njeni pass-ovi) iz primera iznad

RHS Move

RHS Move pass je jednostavan i sastoji se od prebacivanja konstante na desnu stranu komutativnih binarnih operatora (konkretno $+$ i $*$).

Primer:

$$2 * x \rightarrow x * 2$$

$$2 + x \rightarrow x + 2$$

Primer sa LLVM IR:

```
%5 = load i32, ptr %2, align 4
%6 = add nsw i32 2, %5
```

postaje

```
%5 = load i32, ptr %2, align 4
%6 = add i32 %5, 2
```

Drugi primer sa LLVM IR:

```
%7 = load i32, ptr %2, align 4
%8 = mul nsw i32 2, %7
```

postaje

```
%7 = load i32, ptr %2, align 4
%8 = mul i32 %7, 2
```

Convert compares

Convert compares pass pretvara operacije poređenja (<, >, <=, >=) u == ili !=, ukoliko je to moguće.

Primer LLVM IR (Pretvara >= u !=):

```
store i32 3, i32* %3, align 4
%4 = load i32, i32* %2, align 4
%5 = icmp sgt i32 %4, 0
br i1 %5, label %6, label %9
```

pretvara u:

```
store i32 3, i32* %3, align 4
%4 = load i32, i32* %2, align 4
%5 = icmp ne i32 %4, 0
br i1 %5, label %6, label %9
```

Primer kombinacije LLVM IR (Pretvara \leq u *true*):

```
%16 = load i32, i32* %2, align 4
%17 = icmp sle i32 %16, 0
br i1 %17, label %18, label %19

18:
```

menja u:

```
18:
%19 = load i32, i32* %2, align 4
br i1 true, label %20, label %21

20:
```

Replace compares

Replace compares pass menja operacije poređenja nad bool vrednostima sa logičkim operacijama.

Primer:

$$\begin{aligned}x == y &\rightarrow x \text{ not xor } y \\x != y &\rightarrow x \text{ xor } y\end{aligned}$$

Primer LLVM IR (Pretvaranje `==` u `not xor`):

```
%10 = trunc i64 %9 to i32
%11 = zext i1 %10 to i32
%12 = icmp eq i32 %8, %11
br i1 %12, label %13, label %14
```

pretvara u:

```
%10 = trunc i64 %9 to i32
%11 = zext i1 %10 to i32
%12 = xor i32 %8, %11
%13 = xor i32 %12, -1
br i32 %13, label %14, label %15
```

Drugi primer LLVM IR (Pretvara != u xor):

```
%20 = zext i1 %19 to i32
%21 = icmp ne i32 %17, %20
br i1 %21, label %22, label %23
```

pretvara u:

```
%20 = trunc i8 %19 to i1
%21 = zext i1 %20 to i32
%22 = xor i32 %18, %21
br i32 %22, label %23, label %24
```


Replace same-operand Add with Shift

Replace same-operand Add with Shift je pass koji menja sabiranje, u kom su oba sabirka ista, sa bitovskom Shift operacijom.

Primer:

$$x + x \quad \rightarrow \quad x \ll 1$$

Primer sa LLVM IR:

```
%4 = load i32, i32* %2, align 4
%5 = load i32, i32* %2, align 4
%6 = add nsw i32 %4, %5
store i32 %6, i32* %3, align 4
ret i32 0
```

se menja u:

```
%4 = load i32, i32* %2, align 4
%5 = load i32, i32* %2, align 4
%6 = shl i32 %4, 1
store i32 %6, i32* %3, align 4
ret i32 0
```

Replace powers of 2 with Shift

Replace powers of 2 with shift je pass koji stepene dvojke pretvara u odgovarajuću bitovsku operaciju Shift.

Primeri:

$x * 4 \rightarrow x \ll 2$

$x * 8 \rightarrow x \ll 3$

Primer sa LLVM IR:

```
store i32 2, i32* %2, align 4
%5 = load i32, i32* %2, align 4
%6 = mul nsw i32 %5, 4
store i32 %6, i32* %3, align 4
%7 = load i32, i32* %2, align 4
%8 = mul nsw i32 %7, 8
store i32 %8, i32* %4, align 4
```

se menja u:

```
%5 = load i32, i32* %2, align 4
%6 = shl i32 %5, 2
store i32 %6, i32* %3, align 4
%7 = load i32, i32* %2, align 4
%8 = shl i32 %7, 3
store i32 %8, i32* %4, align 4
```

Optimizacija 1 iz uvoda

Podsećanja radi, ova optimizacija radi sledeće:

$$y = x + 1;$$

$$z = y + 1;$$

menja u

$$z = x + 2;$$

U trenutku izvršavanja ove optimizacije, prethodnih 5 pass-eva je već odradilo svoj deo posla.

Primer sa LLVM IR:

```
%1 = alloca i32, align 4
%2 = alloca i32, align 4
%3 = alloca i32, align 4
%4 = alloca i32, align 4
store i32 0, i32* %1, align 4
store i32 1, i32* %2, align 4
%5 = load i32, i32* %2, align 4
%6 = add nsw i32 %5, 1
store i32 %6, i32* %3, align 4
%7 = load i32, i32* %3, align 4
%8 = add nsw i32 %7, 1
store i32 %8, i32* %4, align 4
ret i32 0
```

se menja u:

```
%1 = alloca i32, align 4
%2 = alloca i32, align 4
%3 = alloca i32, align 4
store i32 0, i32* %1, align 4
store i32 1, i32* %2, align 4
%4 = load i32, i32* %2, align 4
%5 = add i32 %4, 2
ret i32 0
```

Optimizacija 2 iz uvoda

Podsećanja radi, optimizacija 2 iz uvoda se bavi inkrementima, i podržava sledeće promene:

`x++;`

`x++;`

se menja u

`x += 2;`

Dozvoljeno je i mešanje više promenljivih:

`x++;`

`y++;`

`y++;`

`x++;`

se menja u

`x += 2;`

`y += 2;`

Optimizacija 2 takođe koristi dva pass-a specifična za nju:

1) Alloc instruction count

- Broji *alloca* instrukcije na početku
- Pravi jaku pretpostavku da nakon toga sledi isto toliko *store* instrukcija

2) Pattern (load-add-store) count

- Prolazi kroz LLVM IR i beleži koliko puta se pojavljuje increment pattern (load-add-store)

Primer LLVM IR (x++; x++;)

```
define dso_local i32 @main() #0 {  
    %1 = alloca i32, align 4  
    %2 = alloca i32, align 4  
    store i32 0, ptr %1, align 4  
    store i32 0, ptr %2, align 4  
    %3 = load i32, ptr %2, align 4  
    %4 = add nsw i32 %3, 1  
    store i32 %4, ptr %2, align 4  
    %5 = load i32, ptr %2, align 4  
    %6 = add nsw i32 %5, 1  
    store i32 %6, ptr %2, align 4  
    ret i32 0  
}
```

se menja u:

```
define dso_local i32 @main() #0 {  
    %1 = alloca i32, align 4  
    %2 = alloca i32, align 4  
    store i32 0, ptr %1, align 4  
    store i32 0, ptr %2, align 4  
    %3 = load i32, ptr %2, align 4  
    %4 = add i32 %3, 2  
    store i32 %4, ptr %2, align 4  
    ret i32 0  
}
```

Drugi primer (x++; y++; y++; x++;):

```
define dso_local i32 @main() #0 {  
    %1 = alloca i32, align 4  
    %2 = alloca i32, align 4  
    %3 = alloca i32, align 4  
    store i32 0, ptr %1, align 4  
    store i32 0, ptr %2, align 4  
    store i32 0, ptr %3, align 4  
    %4 = load i32, ptr %2, align 4  
    %5 = add nsw i32 %4, 1  
    store i32 %5, ptr %2, align 4  
    %6 = load i32, ptr %3, align 4  
    %7 = add nsw i32 %6, 1  
    store i32 %7, ptr %3, align 4  
    %8 = load i32, ptr %3, align 4  
    %9 = add nsw i32 %8, 1  
    store i32 %9, ptr %3, align 4  
    %10 = load i32, ptr %2, align 4  
    %11 = add nsw i32 %10, 1  
    store i32 %11, ptr %2, align 4  
    ret i32 0  
}
```

se menja u:

```
define dso_local i32 @main() #0 {  
    %1 = alloca i32, align 4  
    %2 = alloca i32, align 4  
    %3 = alloca i32, align 4  
    store i32 0, ptr %1, align 4  
    store i32 0, ptr %2, align 4  
    %4 = load i32, ptr %2, align 4  
    %5 = add i32 %4, 2  
    store i32 %5, ptr %2, align 4  
    store i32 0, ptr %3, align 4  
    %6 = load i32, ptr %3, align 4  
    %7 = add i32 %6, 2  
    store i32 %7, ptr %3, align 4  
    ret i32 0  
}
```

Optimizacija 3 iz uvoda

Podsećanja radi, optimizacija 3 radi sledeće:

```
int x = 2;
```

```
x ++;
```

se menja u:

```
int x = 3;
```

Ova optimizacija se nadovezuje na optimizaciju 2 (i njene specifične pass-eve).

Primer LLVM IR:

```
define dso_local i32 @main() #0 {  
    %1 = alloca i32, align 4  
    %2 = alloca i32, align 4  
    store i32 0, ptr %1, align 4  
    store i32 2, ptr %2, align 4  
    %3 = load i32, ptr %2, align 4  
    %4 = add i32 %3, 1  
    store i32 %4, ptr %2, align 4  
    ret i32 0  
}
```

se menja u:

```
define dso_local i32 @main() #0 {  
    %1 = alloca i32, align 4  
    %2 = alloca i32, align 4  
    store i32 0, ptr %1, align 4  
    store i32 3, ptr %2, align 4  
    ret i32 0  
}
```

Primer zajedničkog rada optimizacije 2 i 3:

```
int main() {  
    int x = 2;  
    int y = 0;  
  
    x += 2;  
    x++;  
  
    y++;  
  
    return 0;  
}
```

Neoptimizovani LLVM IR ekvivalentan primeru iznad:

```
define dso_local i32 @main() #0 {  
    %1 = alloca i32, align 4  
    %2 = alloca i32, align 4  
    %3 = alloca i32, align 4  
    store i32 0, ptr %1, align 4  
    store i32 2, ptr %2, align 4  
    store i32 0, ptr %3, align 4  
    %4 = load i32, ptr %2, align 4  
    %5 = add nsw i32 %4, 2  
    store i32 %5, ptr %2, align 4  
    %6 = load i32, ptr %2, align 4  
    %7 = add nsw i32 %6, 1  
    store i32 %7, ptr %2, align 4  
    %8 = load i32, ptr %3, align 4  
    %9 = add nsw i32 %8, 1  
    store i32 %9, ptr %3, align 4  
    ret i32 0  
}
```

Optimizovani LLVM IR ekvivalentan primeru iznad:

```
define dso_local i32 @main() #0 {  
    %1 = alloca i32, align 4  
    %2 = alloca i32, align 4  
    %3 = alloca i32, align 4  
    store i32 0, ptr %1, align 4  
    store i32 5, ptr %2, align 4  
    store i32 1, ptr %3, align 4  
    ret i32 0  
}
```

Function main is valid!

