Projekat: Konstrukcija kompilatora

Marko Nikitovic 123/2020

Tamara Stojanovic 85/2019

Tail call elimination

Primer optimizacije:

```
// An example of tail recursive function
void print(int n)
{
    if (n < 0)
       return;
    cout << " " << n;

    // The last executed statement is recursive call
    print(n-1);
}
```

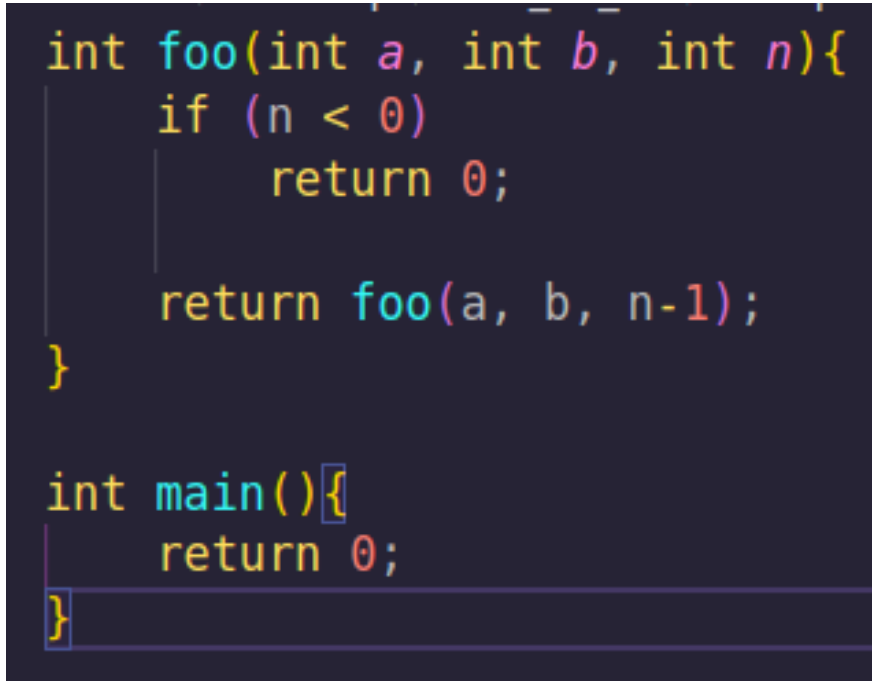
```
// Above code after tail call elimination
void print(int n)
{
start:
    if (n < 0)
       return;
    cout << " " << n;

    // Update parameters of recursive call
    // and replace recursive call with goto
    n = n-1
    goto start;
}
```

Koraci algoritma:

1. Trazimo repno rekuzivan poziv na istu funkciju. Pokriveni slucajevi:
   a. Sledeca instrukcija posle rekurzivnog poziva je **return**
   b. Posle poziva imamo bezuslovni skok na blok koji sadrzi samo **return**
   c. Slucaj kada funkcija nije void, I imamo dodatne **store** i **load** instrukcije
2. Ukoliko nije kreiran, kreiramo blok na koji cemo se vracati i u njega smestamo sve instrukcije iz 'entry' bloka osim **alloca** i **store** instrukcija koji dodeljuju parametre memorijskim lokacijama. Takodje vrsimo preslikavanje izmedju parametara i njihovih memorijskih lokacija. Dodajemo i bezuslovni skok iz entry bloka na vec pomenuti blok.
3. Umesto poziva funkcije kreiramo bezuslovni **br** na novi blok, poziv brisemo a argumente cuvamo (**store**) na preslikanim memorijskim lokacijama

Primenicemo optimizaciju na sledecem kodu u C-u.



```c
int foo(int a, int b, int n){
    if (n < 0)
        return 0;

    return foo(a, b, n-1);
}

int main(){
    return 0;
}
```

```
define dso_local i32 @foo(i32 noundef %0, i32 noundef %1, i32 noundef %2) #0
  %4 = alloca i32, align 4
  %5 = alloca i32, align 4
  %6 = alloca i32, align 4
  %7 = alloca i32, align 4
  store i32 %0, ptr %5, align 4
  store i32 %1, ptr %6, align 4
  store i32 %2, ptr %7, align 4
  %8 = load i32, ptr %7, align 4
  %9 = icmp slt i32 %8, 0
  br i1 %9, label %10, label %11

10:                                               ; preds = %3
  store i32 0, ptr %4, align 4
  br label %17

11:                                               ; preds = %3
  %12 = load i32, ptr %5, align 4
  %13 = load i32, ptr %6, align 4
  %14 = load i32, ptr %7, align 4
  %15 = sub nsw i32 %14, 1
  %16 = call i32 @foo(i32 noundef %12, i32 noundef %13, i32 noundef %15)
  store i32 %16, ptr %4, align 4
  br label %17

17:                                               ; preds = %11, %10
  %18 = load i32, ptr %4, align 4
  ret i32 %18
}
```

```
; Function Attrs: noinline nounwind optnone uwtable
define dso_local i32 @foo(i32 noundef %0, i32 noundef %1, i32 noundef %2) #0
  %4 = alloca i32, align 4
  %5 = alloca i32, align 4
  %6 = alloca i32, align 4
  %7 = alloca i32, align 4
  store i32 %0, ptr %5, align 4
  store i32 %1, ptr %6, align 4
  store i32 %2, ptr %7, align 4
  br label %loop_start

loop_start:                                       ; preds = %11, %3
  %8 = load i32, ptr %7, align 4
  %9 = icmp slt i32 %8, 0
  br i1 %9, label %10, label %11

10:                                               ; preds = %loop_start
  store i32 0, ptr %4, align 4
  br label %16

11:                                               ; preds = %loop_start
  %12 = load i32, ptr %5, align 4
  %13 = load i32, ptr %6, align 4
  %14 = load i32, ptr %7, align 4
  %15 = sub nsw i32 %14, 1
  store i32 %12, ptr %5, align 4
  store i32 %13, ptr %6, align 4
  store i32 %15, ptr %7, align 4
  br label %loop_start

16:                                               ; preds = %10
  %17 = load i32, ptr %4, align 4
  ret i32 %17
}
```

Copy propagation

```cpp
// Example for Local Copy Propagation
// Before applying Copy Propagation
#include <iostream>
using namespace std;

int main()
{

    int a = 1 + 2;
    int b = a;
    int ans = b + 6;
    cout << "Before copy propagation, ans= " << ans;

    return 0;
}
```

```cpp
// Example for Local Copy Propagation
// After applying Copy Propagation
#include <iostream>
using namespace std;

int main()
{

    int a = 1 + 2;
    int ans = a + 6;
    cout << "After copy propagation, ans= " << ans;

    return 0;
}
```

1. Prvo vrsimo mapiranje promenljivih na odgovarajuce memorijske lokacije
2. Za svaki blok racunamo CPIn i CPOut
   a. CPIn[BB] skup kopija dostupnih na ulazu bloka BB

$$CPin(i) = \bigcap_{j \in Pred(i)} CPout(j)$$

   b. CPOut[BB] skup kopija dostupnih na izlazu bloka

$$CPout(i) = COPY(i) \cup (CPin(i) - KILL(i))$$

3. Za svaki blok onda vrsimo lokalnu propagaciju
   a. Skup kopija za blok B1 postavljamo na CPIn[B1]
   b. Prolazimo kroz skup instrukcija I gledamo da li su **load** ili **store**
   c. Ukoliko je store proveravamo da li je potrebno odraditi update skupa kopija
   d. Ukoliko je load, pokusavamo da zamenimo pointer operand

Primer:

```c
int main(){
    int a, b, c;

    a = 2;
    b = a;

    if (c > 5){
        b = 3;
    }
    else{
        printf("%d", b);
    }

    while (c == 5){
        printf("%d", b);
    }


    return 0;
}
```

```
define dso_local i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4
  %4 = alloca i32, align 4
  store i32 0, ptr %1, align 4
  store i32 2, ptr %2, align 4
  %5 = load i32, ptr %2, align 4
  store i32 %5, ptr %3, align 4
  %6 = load i32, ptr %4, align 4
  %7 = icmp sgt i32 %6, 5
  br i1 %7, label %8, label %9

8:                                                ; preds = %0
  store i32 3, ptr %3, align 4
  br label %12

9:                                                ; preds = %0
  %10 = load i32, ptr %3, align 4
  %11 = call i32 (ptr, ...) @printf(ptr noundef @.str, i32 noundef %10)
  br label %12

12:                                               ; preds = %9, %8
  br label %13

13:                                               ; preds = %16, %12
  %14 = load i32, ptr %4, align 4
  %15 = icmp eq i32 %14, 5
  br i1 %15, label %16, label %19

16:                                               ; preds = %13
  %17 = load i32, ptr %3, align 4
  %18 = call i32 (ptr, ...) @printf(ptr noundef @.str, i32 noundef %17)
  br label %13, !llvm.loop !6

19:                                               ; preds = %13
  ret i32 0
}
```

```
define dso_local i32 @main() #0 {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4
  %4 = alloca i32, align 4
  store i32 0, ptr %1, align 4
  store i32 2, ptr %2, align 4
  %5 = load i32, ptr %2, align 4
  store i32 %5, ptr %3, align 4
  %6 = load i32, ptr %4, align 4
  %7 = icmp sgt i32 %6, 5
  br i1 %7, label %8, label %9

8:                                                ; preds = %0
  store i32 3, ptr %3, align 4
  br label %12

9:                                                ; preds = %0
  %10 = load i32, ptr %2, align 4
  %11 = call i32 (ptr, ...) @printf(ptr noundef @.str, i32 noundef %10)
  br label %12

12:                                               ; preds = %9, %8
  br label %13

13:                                               ; preds = %16, %12
  %14 = load i32, ptr %4, align 4
  %15 = icmp eq i32 %14, 5
  br i1 %15, label %16, label %19

16:                                               ; preds = %13
  %17 = load i32, ptr %3, align 4
  %18 = call i32 (ptr, ...) @printf(ptr noundef @.str, i32 noundef %17)
  br label %13, !llvm.loop !6

19:                                               ; preds = %13
  ret i32 0
}
```

Literatura:

1. Advanced Compiler Design and Implementation, S. S. Muchnick
2. https://groups.seas.harvard.edu/courses/cs153/2019fa/lectures/Lec19-Optimization.pdf
3. https://www.geeksforgeeks.org/