CFG Loop con flag -O2

```
source_filename = "test/Loop.c"
target datalayout = "e-m:o-p270:32:32-p271:32:32-p272:64:64-164:64-f80:128-n8:16:32:64-S128"
target triple = "x86_64-apple-macosx12.0.0"
@g = local_unnamed_addr global i32 0, align 4
Function Attrs: mustprogress nofree norecurse nosync nounwind ssp willreturn uwtable
define i32 @g_incr(i32 noundef %0) local_unnamed_addr #0 {
  %2 = load i32, ptr @g, align 4, !tbaa !5
  %3 = add nsw i32 %2, %0
                                                                  001
  store i32 %3, ptr @g, align 4, !tbaa !5
  ret i32 %3
 Function Attrs: nofree norecurse nosync nounwind ssp uwtable
define i32 @loop(i32 noundef %0, i32 noundef %1, i32 noundef %2) local_unnamed_addr #1 {
  %4 = load i32, ptr @g, align 4, !tbaa !5
  %5 = icmp sgt i32 %1, %0
                                                                      BBZ
    i1 %5, label %6, label %10
                                                 ; preds = %3
  %7 = sub i32 %1, %0
  %8 = mul i32 %7, %2
  %9 = add i32 %4, %8
                                                     133
  store i32 %9, ptr @g, align 4, !tbaa !5
  br label %10
                                                 ; preds = %6, %3
  %11 = phi i32 [ %9, %6 ], [ %4, %3 ]
  ret i32 %11
                                                       DB4
```

Il compilatore elimina la chiamata diretta alla funezione g_incr() e l'iterazione del for sostituendole con operazioni algebriche.

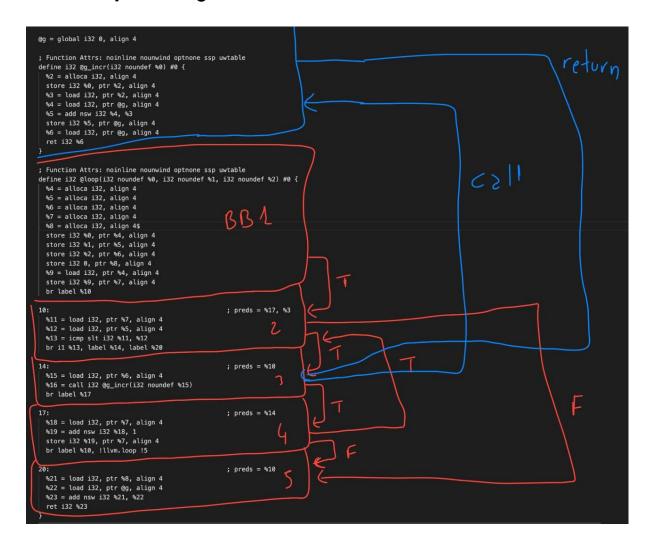
In particlare si calcola algebricamente il numero di cicli del for facendo una sottrazione tra la variabile B e la variabile A. Dopodichè aggiunge alla variabile RET tante volte C quante sono i cicli da eseguire. Per ottimizare questa istruzione esegue direttamente una moltiplicazione tra la il numero di cicli (B-A) e il valore di C.

In questo modo elimina le diverse chiamate alla funzione g_inc() ed elimina il for.

Inoltre non considera l'allocazione della variabile I e della variabile RET, in quanto la variabile I viene assegnata al valore di A e la variabile ret viene considerata come la variabile globale G.

L'assegnamento finale della variabile RET e G non causa problemi perchè la variabile G è globale ed ha valore 0, alla stesso tempo anche RET viene inizializzata al valore 0. Di conseguenza la somma delle due può essere eliminata e lavorare solo con una variabile.

CFG Loop con flag -O0



In questo caso il compilatore non esegue nessun tipo di ottimizzazione.

In particolare procede all'allocazione di tutte le variabili, all'esecuzione completa del FOR e alla chiamata diretta alla funzione g_inc().

CFG Fibonacci con flag -O2

```
Function Attrs: mustprogress nofree nounwind ssp uwtable
  define i32 @printf(ptr nocapture noundef readonly %0, ...) local_unnamed_addr #0 {
%2 = alloca [1 x %struct.__va_list_tag], align 16
call void @llvm.lifetime.start.p0(i64 24, ptr nonnull %2) #4
                                                                                                                                             DET
                                                                                                                                                                                    160
   call void @llvm.va_start(ptr nonnull %2)
%3 = load ptr, ptr @_stdoutp, align 8, !tbaa !5
%4 = call i32 @vfprintf(ptr noundef %3, ptr noundef %0, ptr noundef nonnull %2)
call void @llvm.va_end(ptr nonnull %2)
                                                                                                                                              DFT
                                                                                                                                                                                       1150
                                                                                                                                                                                       return
                                                                                                                                     4
     call void @llvm.lifetime.end.p0(i64 24, ptr nonnull %2) #4
                                                                                                                                                DFT
                                                                                                                                                                                   150
                                                                                                                                    5
    ret i32 %4
 ; Function Attrs: mustprogress nofree nounwind ssp uwtable
 define noundef i32 @_Z9Fibonaccii(i32 noundef %0) local_unnamed_addr #0 {
   br label %2
                                                                              ; preds = %5, %1
    :

%3 = phi i32 [ 0, %1 ], [ %10, %5 ]

%4 = phi i32 [ %0, %1 ], [ %7, %5 ]

switch i32 %4, label %5 [

i32 0, label %12

i32 1, label %11
                                                                                                                                                     JFT (?)
                                                                                                                                                                                                        call
    %6 = add nsw i32 %4, -1
    %7 = add nsw i32 %4, -2
%7 = add nsw i32 %4, -2
%8 = tail call i32 (ptr, ...) @printf(ptr noundef nonnull @.str.2, i32 noundef %4, i32 noundef %6, i32 noundef %7)
%9 = tail call noundef i32 @_Z9Fibonaccii(132 noundef %6)
    %10 = add nsw 132 %9, %3
br label %2
                                                                                                                                                                                                            ret
                                                                              : preds = %2
    br label %12
                                                                              ; preds = %2, %11
   %13 = phi ptr [ @.str.1, %11 ], [ @.str, %2 ]
%14 = tail call i32 (ptr, ...) @printf(ptr noundef nonnull %13)
%15 = add nsw i32 %4, %3
                                                                                                                                                                                                   return
    ret i32 %15
 ; Function Attrs: argmemonly mustprogress nocallback nofree nosync nounwind willreturn
 declare void @llvm.lifetime.start.p0(i64 immarg, ptr nocapture) #1
 ; Function Attrs: mustprogress nocallback nofree nosync nounwind willreturn
 declare void @llvm.va_start(ptr) #2
 ; Function Attrs: nofree nounwind
 declare noundef i32 @vfprintf(ptr nocapture noundef, ptr nocapture noundef readonly, ptr noundef) local_unnamed_addr #3
 ; Function Attrs: mustprogress nocallback nofree nosync nounwind willreturn declare void @llvm.va_end(ptr) \#2
 ; Function Attrs: argmemonly mustprogress nocallback nofree nosync nounwind willreturn
  declare void @llvm.lifetime.end.p0(i64 immarg, ptr nocapture) #1
; Function Attrs: mustprogress noinline optnone ssp uwtable
; Function Attrs: mustprogress noinline optnone ssp uwtable
define i32 @printf(ptr noundef %0, ...) #0 {
%2 = alloca ptr, align 8
%3 = alloca i32, align 4
%4 = alloca [1 x %struct.__va_list_tag], align 16
store ptr %0, ptr %2, align 8
%5 = getelementptr inbounds [1 x %struct.__va_list_tag], ptr %4, i64 0, i64 0
call void @llvm.va_start(ptr %5)
%5 = load str. vis. 2 stdext align 8
                                                                                                                                                  veturn
   %6 = load ptr, ptr @_stdoutp, align 8
%7 = load ptr, ptr %2, align 8
%8 = getelementptr inbounds [1 x %struct.__va_list_tag], ptr %4, i64 0, i64 0
%9 = call i32 @vfprintf(ptr noundef %6, ptr noundef %7, ptr noundef %8)
   %10 = getelementptr inbounds [1 x %struct._va_list_tag], ptr %4, i64 0, i64 0 call void @llvm.va_end(ptr %10)
                                                                                                                                                                  6311
                                                                                                                                        return
   %11 = load i32, ptr %3, align 4
   ret i32 %11
```

In questo la funzione di Fibonacci viene modificata inserendo al posto delle chiamate normali delle TAIL CALL che permettono di ottimizzare l'esecuzione complessiva del codice.

CFG Fibonacci con flag -O0

```
; Function Attrs: noinline nounwind optnone ssp uwtable
define i32 @Fibonacci(i32 noundef %0) #0 {
                                                                     BB1
  %2 = alloca i32, align 4
  %3 = alloca i32, align 4
  store i32 %0, ptr %3, align 4
  %4 = load i32, ptr %3, align 4
  %5 = icmp eq i32 %4, 0
  br i1 %5, label %6, label %8
                                                                                                                                                    10101
                                                               ; preds = %1
%7 = call i32 (ptr, ...) @printf(ptr noundef @.str)
  store i32 0, ptr %2, align 4
  br label %27
                                                                                                                                             C911
  %9 = load i32, ptr %3, align 4
  %10 = icmp eq i32 %9, 1
  br i1 %10, label %11, label %13
                                                               ; preds = %8
 %12 = call i32 (ptr, ...) @printf(ptr noundef @.str.1)
  store i32 1, ptr %2, align 4
br label %27
                                                               ; preds = %8
  %14 = load i32, ptr %3, align 4
  %15 = load i32, ptr %3, align 4
  %16 = sub nsw i32 %15, 1
  %17 = load i32, ptr %3, align 4
  %18 = sub nsw i32 %17, 2
  %19 = call i32 (otr, ...) @printf(ptr noundef @.str.2, i32 noundef %14, i32 noundef %16, i32 noundef %18)
%20 = load i32, ptr %3, align 4
                                                                                                                                                           E511
  %21 = sub nsw i32 %20, 1
  %22 = call i32 @Fibonacci(i32 noundef %21)
                                                                                                                                                return
  %23 = load i32, ptr %3, align 4
                                                                                                                         9
  %24 = sub nsw i32 %23, 2
  %25 = call i32 @Fibonacci(i32 noundef %24)
  %26 = add nsw i32 %22, %25
                                                                                                                            10
  store i32 %26, ptr %2, align 4
                                                                                                                                                       return
  br label %27
                                                               ; preds = %13, %11, %6
 %28 = load i32, ptr %2, align 4
                                                                                                                              11
  ret i32 %28
  oot@PC-Davide:/home/dave/programmazione/Linguaggi/Maroungiu/Linguaggi-e-Compilatori-2022-2023/Tutorial-01/TestPass# clang -O2 -Rpass=
ci.c -o test/Fibonacci.ll

test/Fibonacci.c:24:29: remark: transforming tail recursion into loop [-Rpass=tailcallelim]

return Fibonacci(n - 1) + Fibonacci(n - 2);
test/Fibonacci.c:24:29: remark: advising against unrolling the loop because it contains a call [-Rpass=TTI]
test/Fibonacci.c:24:29: remark: advising against unrolling the loop because it contains a call [-Rpass=TTI]
root@PC-Davide:/home/dave/programmazione/Linguaggi/Maroungiu/Linguaggi-e-Compilatori-2022-2023/Tutorial-01/TestPass# clang -02 -Rpass=.* -emit
-o test/Loop.ll
test/Loop.c:34:5: remark: 'g_incr' inlined into 'loop' with (cost=-20, threshold=337) at callsite loop:4:5; [-Rpass=inline] g_incr(c);
test/Loop.c:21:5: remark: Moving accesses to memory location out of the loop [-Rpass=licm]
test/Loop.c:21:5: remark: Moving accesses to memory location out of the loop [-Rpass=licm] test/Loop.c:33:3: remark: Loop deleted because it is invariant [-Rpass=loop-delete] for (i = a; i < b; i++) {
test/Loop.c:37:16: remark: load of type i32 eliminated [-Rpass=gvn]
  return ret + g;
```