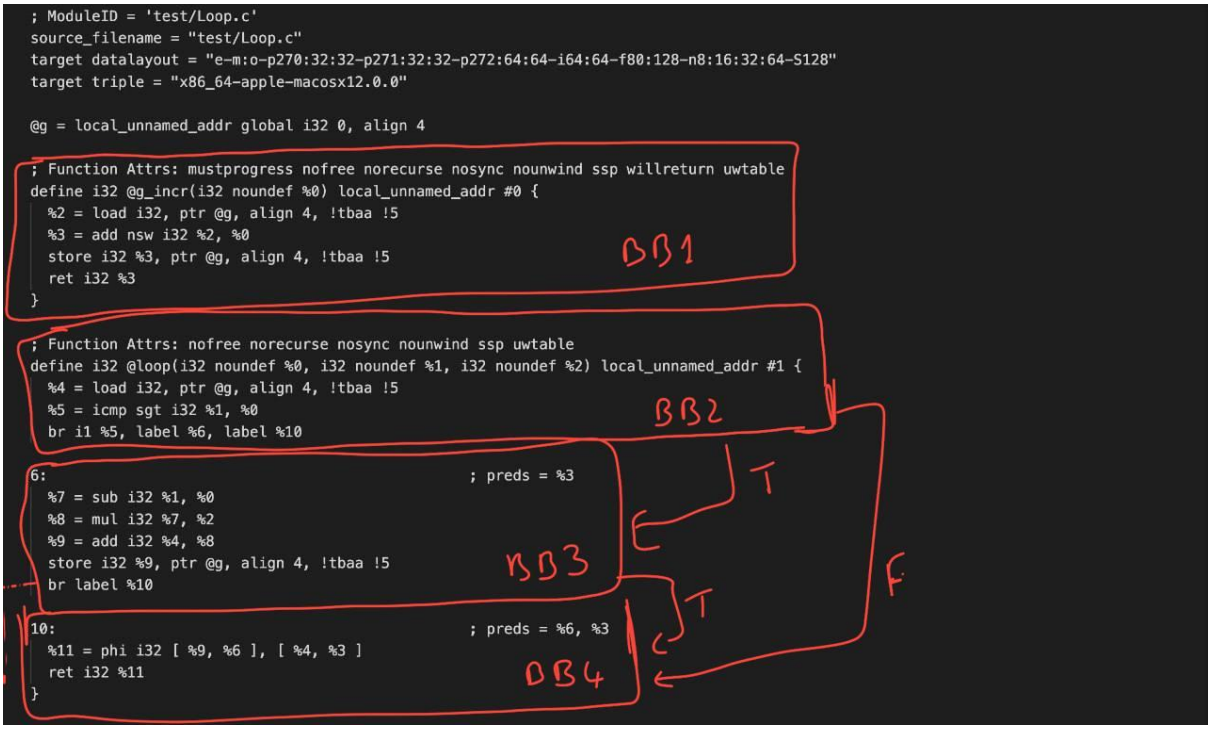
CFG Loop con flag -O2

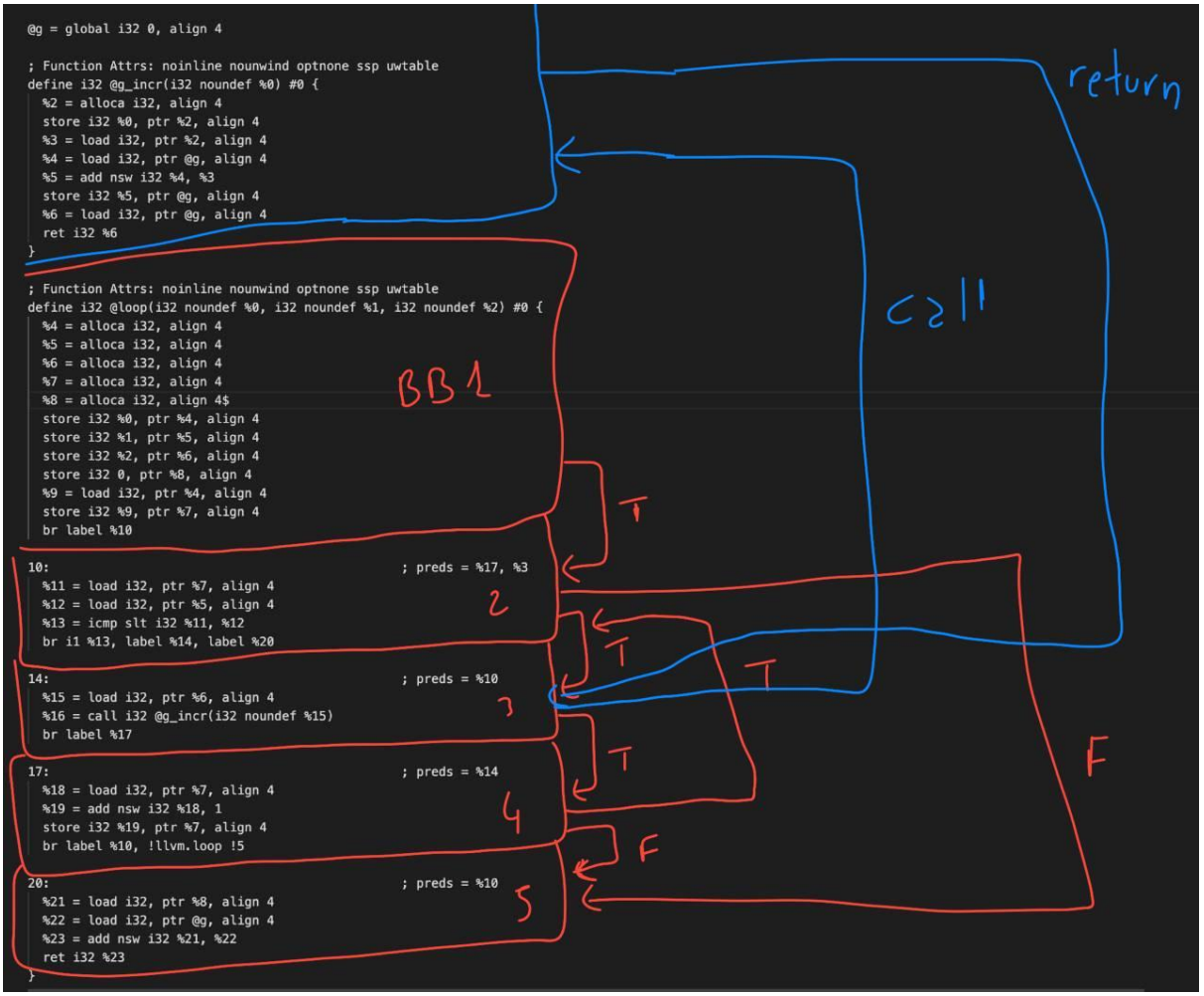


Il compilatore elimina la chiamata diretta alla funezione g_incr() e l'iterazione del for sostituendole con operazioni algebriche.

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.

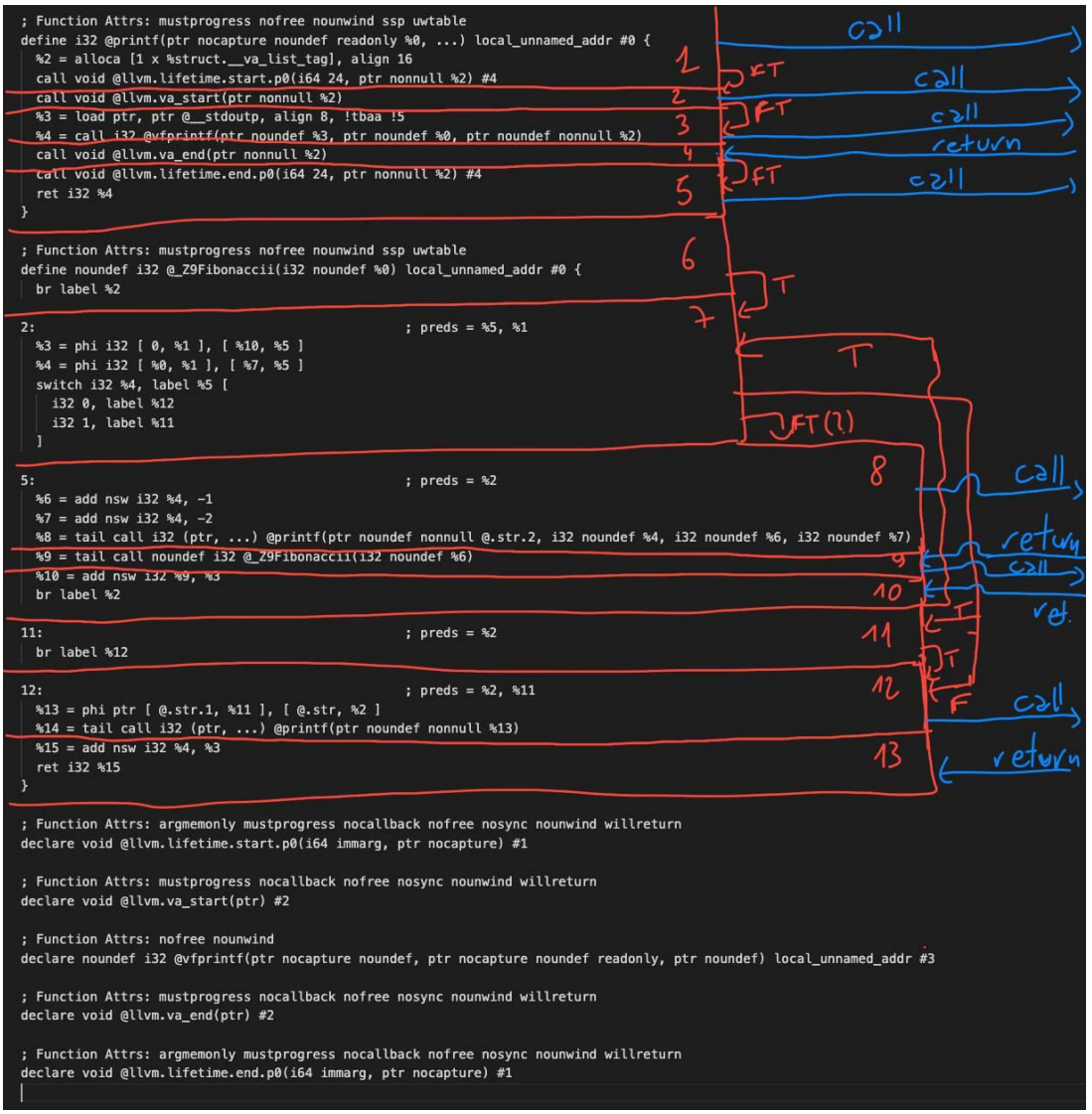
CFG Loop con flag -O0



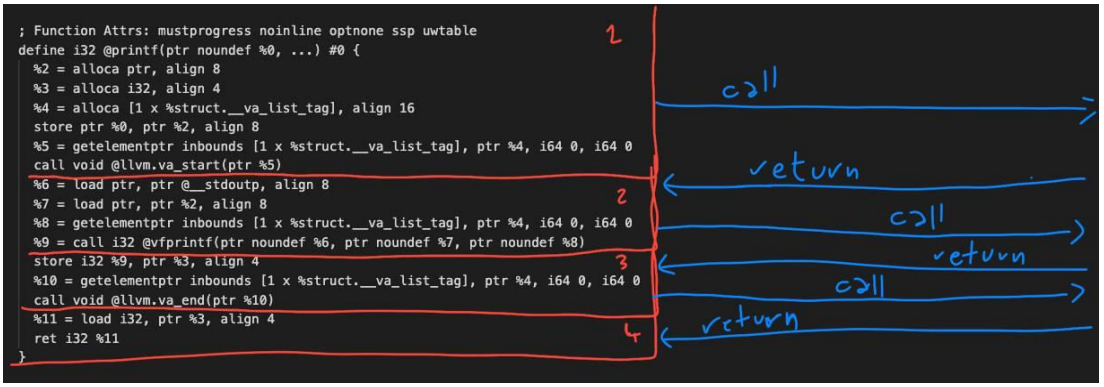
In questo caso il compilatore non esegue nessun tipo di ottimizzazione.

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
  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)
  call void @llvm.lifetime.end.p0(i64 24, ptr nonnull %2) #4
  ret i32 %4
}

; Function Attrs: mustprogress nofree nounwind ssp uwtable
define noundef i32 @_Z9Fibonaccii(i32 noundef %0) local_unnamed_addr #0 {
  br label %2

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
  ]

5:                                                ; preds = %2
  %6 = add nsw i32 %4, -1
  %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(i32 noundef %6)
  %10 = add nsw i32 %9, %3
  br label %2

11:                                               ; preds = %2
  br label %12

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
  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
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)
  %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)
  store i32 %9, ptr %3, align 4
  %10 = getelementptr inbounds [1 x %struct.__va_list_tag], ptr %4, i64 0, i64 0
  call void @llvm.va_end(ptr %10)
  %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


```
root@PC-Davide:/home/dave/programmazione/Linguaggi/Maroungiu/Linguaggi-e-Compilatori-2022-2023/Tutorial-01/TestPass# clang -O2 -Rpass=.* -emit
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 -O2 -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]
  g += c;
  ^

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;
             ^
```