

# Relazione lavoro svolto

Pietro Ghiglio

September 18, 2020

# Table of Contents

- 1 Instrumentazione
  - Funzioni ricorsive
- 2 Metriche
- 3 Mapping Source  $\rightarrow$  LLVM  $\rightarrow$  Assembly
- 4 Demo
- 5 Conclusioni

# Table of Contents

- 1 **Instrumentazione**
  - Funzioni ricorsive
- 2 Metriche
- 3 Mapping Source  $\rightarrow$  LLVM  $\rightarrow$  Assembly
- 4 Demo
- 5 Conclusioni

# Scopo

- Propagare a livello di codice sorgente informazioni contenute tramite profiling.
- Associare ad ogni riga di codice il numero di istruzioni provenienti da essa eseguite durante la run del programma.
- Associare ad ogni chiamata a funzione il numero di istruzioni eseguite a partire dalla chiamata.

# Implementazione

- Stack di callsites, push prima di ogni chiamata, pop dopo la chiamata.
- Ad ogni esecuzione di un basic block: stampa id basic block + dump dello stack.

# Propagazione a livello di codice sorgente

Dato un output dell'istrumentazione corrispondente all'esecuzione di un basic block: `bbld callsite1 callsite2 ... callsiteN`

- Per ogni istruzione llvm contenuta nel basic block corrispondente all'id, assegnare il costo dell'esecuzione dell'istruzione llvm alla location di codice sorgente corrispondente.
- Per ogni callsite nel dump dello stack, assegnare il costo dell'istruzione llvm al callsite, a meno di funzioni ricorsive.

# Call Graph

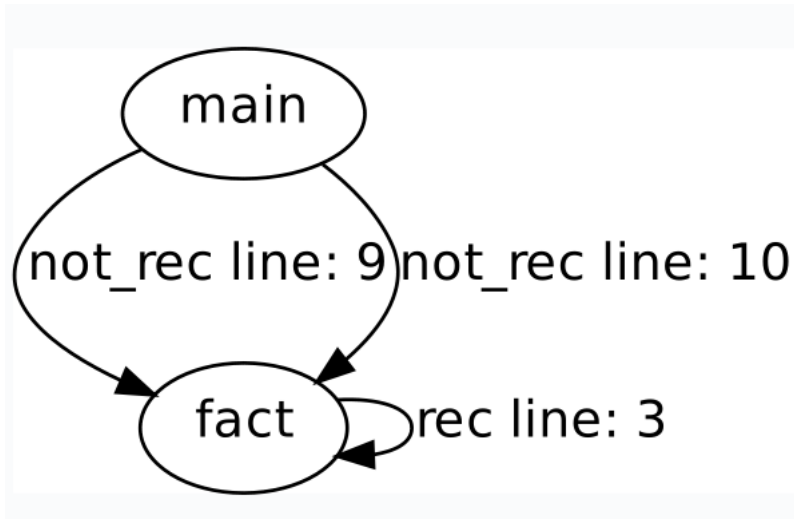
- Grafo in cui ogni vertice è una funzione.
- Un edge  $(v,u)$  rappresenta il fatto che la funzione  $v$  chiama  $u$ .
- Label sugli edge con callsite.
- Una call  $(v,u)$  è ricorsiva se l'edge corrispondente è parte di un ciclo.

# Esempio-1

```
1 unsigned long fact(int n){  
2     if(n <= 1) return 1;  
3     unsigned long a = fact(n-1);  
4     unsigned long b = n * a;  
5     return n*a;  
6 }  
7  
8 int main(){  
9     fact(9);  
10    fact(3);  
11 }
```



## Esempio-1



## Esempio-2

```
1 int f(int n);  
2 int g(int n);  
3 int h(int n);  
4  
5 int main(){  
6     f(3);  
7     g(2);  
8     h(1);  
9 }
```

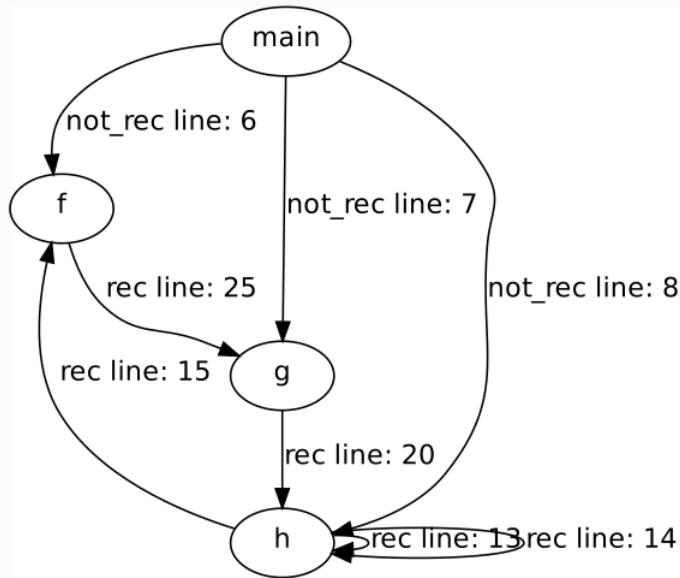
## Esempio-2

```
11 int h(int n){  
12     if(n < 1) return 1;  
13     int a = h(n-1);  
14     int b = h(n-2);  
15     int c = f(a/b);  
16     return c;  
17 }  
18  
19 int g(int n){  
20     int a = h(n-1);  
21     return a;  
22 }
```

## Esempio-2

```
24 int f(int n){  
25     int a = g(n-1);  
26     return a;  
27 }
```

# Esempio-1



# Criterio di attribuzione

Costo dell'istruzione attribuito al callsite solo se non è una chiamata ricorsiva.

# Table of Contents

- 1 Instrumentazione
  - Funzioni ricorsive
- 2 **Metriche**
- 3 Mapping Source  $\rightarrow$  LLVM  $\rightarrow$  Assembly
- 4 Demo
- 5 Conclusioni

# Metriche utilizzate

Al momento le metriche utilizzabili sono:

- Il numero di istruzioni LLVM: ogni istruzione LLVM ha costo 1.
- Il numero di istruzioni Assembly: ogni istruzione LLVM ha costo pari al numero di istruzioni assembly ad essa associato.

Ad entrambe potrebbe essere associato un costo energetico o diretto (istruzioni LLVM) o dato dalla somma del costo delle istruzioni assembly corrispondenti.

Richiede un energy model della target architecture, con le varie considerazioni sulla fattibilità in base alla complessità dell'architettura.



# Table of Contents

- 1 Instrumentazione
  - Funzioni ricorsive
- 2 Metriche
- 3 Mapping Source  $\rightarrow$  LLVM  $\rightarrow$  Assembly
- 4 Demo
- 5 Conclusioni

# Mapping Source → LLVM

Mapping source → LLVM direttamente dalle debug information delle API LLVM (classi `DebugLoc`, `DISubProgram`).

Alcune istruzioni (es. `alloca` all'inizio della funzione) non hanno debug info. In tal caso vengono attribuite alla line di definizione della funzione.

# Mapping LLVM → Assembly

Mapping LLVM → Assembly ottenuto tramite un pass che sostituisce le informazioni di debug riguardo alla linea di codice sorgente con un id dell'istruzione stessa.

Molti metodi delle API LLVM per la modifica delle informazioni di debug sono privati.

Le informazioni di debug sostituite vengono recuperate effettuando disassembly dell'eseguibile.

# Table of Contents

- 1 Instrumentazione
  - Funzioni ricorsive
- 2 Metriche
- 3 Mapping Source  $\rightarrow$  LLVM  $\rightarrow$  Assembly
- 4 Demo
- 5 Conclusioni

# Codice

```
1 unsigned long fact(int n){  
2     if(n <= 1) return 1;  
3     unsigned long a = fact(n-1);  
4     unsigned long b = n * a;  
5     return n*a;  
6 }  
7  
8 int main(){  
9     fact(9);  
10    fact(3);  
11 }
```

# CallGraph

```
1 digraph cg {  
2   fact  
3   main  
4   fact -> fact[label = "rec_line: 3"]  
5   main -> fact[label = "not_rec_line: 9"]  
6   main -> fact[label = "not_rec_line: 10"]  
7 }
```

# Disassembly pre fix

name: main

begin: 401170 end: 40119e

addr: 401170 pushq %rbp debug: 8

addr: 401171 movq %rsp, %rbp

addr: 401174 subq \$16, %rsp

addr: 401178 movl \$9, %edi debug: 31

addr: 40117d callq -114

addr: 401182 movl \$3, %edi debug: 32

addr: 401187 movq %rax, -8(%rbp)

addr: 40118b callq -128

addr: 401190 xorl %ecx, %ecx

addr: 401192 movq %rax, -16(%rbp)

addr: 401196 movl %ecx, %eax debug: 33

addr: 401198 addq \$16, %rsp

addr: 40119c popq %rbp

addr: 40119d retq

addr: 40119e nop debug: 33

# Disassembly post fix

name: main

begin: 401170 end: 40119e

addr: 401170 pushq %rbp debug: 31

addr: 401171 movq %rsp, %rbp debug: 31

addr: 401174 subq \$16, %rsp debug: 31

addr: 401178 movl \$9, %edi debug: 31

addr: 40117d callq -114 debug: 31

addr: 401182 movl \$3, %edi debug: 32

addr: 401187 movq %rax, -8(%rbp) debug: 32

addr: 40118b callq -128 debug: 32

addr: 401190 xorl %ecx, %ecx debug: 32

addr: 401192 movq %rax, -16(%rbp) debug: 32

addr: 401196 movl %ecx, %eax debug: 33

addr: 401198 addq \$16, %rsp debug: 33

addr: 40119c popq %rbp debug: 33

addr: 40119d retq debug: 33

addr: 40119e nop debug: 33



# Istruzioni LLVM

```
31:    %1 = call i64 @fact(i32 9), !dbg !11
32:    %2 = call i64 @fact(i32 3), !dbg !12
33:    ret i32 0, !dbg !13
```

# Attribuzione - LLVM

```
1 unsigned long fact(int n){ //72 llvm instr
2     if(n <= 1) return 1; //40 llvm instr
3     unsigned long a = fact(n-1); //50 llvm instr
4     unsigned long b = n * a; //60 llvm instr
5     return n*a; //60 llvm instr
6 } //24 llvm instr
7
8 int main(){
9     fact(9); //238 llvm instr
10    fact(3); //70 llvm instr
11 } //1 llvm instr
```

# Attribuzione - Assembly

```
1 unsigned long fact(int n){ //48 assembly instr
2     if(n <= 1) return 1; //28 assembly instr
3     unsigned long a = fact(n-1); //50 assembly instr
4     unsigned long b = n * a; //30 assembly instr
5     return n*a; //30 assembly instr
6 } //60 assembly instr
7
8 int main(){
9     fact(9); //194 assembly instr
10    fact(3); //62 assembly instr
11 } //5 assembly instr
```

# Table of Contents

- 1 Instrumentazione
  - Funzioni ricorsive
- 2 Metriche
- 3 Mapping Source  $\rightarrow$  LLVM  $\rightarrow$  Assembly
- 4 Demo
- 5 Conclusioni

# Come procedere ora?

- Function mangling.
- File multipli.
- Energy model.
- Ottimizzazioni del compilatore.
- Chiamate a libreria.