Corso di Laurea Magistrale in Ingegneria Informatica
A.A. 2013-2014

Linguaggi Formali e Compilatori

Generazione/Ottimizzazione del codice (cenni)

Giacomo PISCITELLI

Generazione del codice

La fase finale del compilatore è quella della generazione del codice e conseguente sua ottimizzazione. In tale fase viene preso il codice nella sua rappresentazione intermedia (IR), insieme con le informazioni contenute nella *symbol table* e viene prodotto un programma target semanticamente equivalente.

I compilatori più sofisticati effettuano più passi di elaborazione della IR. Ciò è dovuto alla maggiore facilità di applicazione, uno alla volta, degli algoritmi di ottimizzazione o al fatto che l'input ad una ottimizzazione dipende dall'output prodotto da un'altra ottimizzazione.

Inoltre questa struttura del processo di generazione del codice facilita la creazione di un singolo compilatore che può sfruttare più architetture, in quanto solo l'ultimo stadio di generazione del codice necessita dei cambiamenti dipendenti dalla macchina target.

Requisiti del generatore di codice

I requisiti a cui deve ottemperare un generatore di codice sono stringenti e severi: il programma generato deve preservare il significato semantico del programma sorgente ed essere di qualità; deve, cioè, fare un uso efficace delle risorse della macchina target. Il generatore di codice stesso deve essere efficiente.

La sfida nasce dal fatto che, matematicamente, il problema di generare un codice target ottimale per un dato programma sorgente è indecidibile: molti dei sottoproblemi posti dalla generazione del codice, come ad esempio l'allocazione dei registri, sono computazionalmente intrattabili.

In pratica bisogna accontentarsi delle tecniche euristiche per produrre un buon codice, anche se non ottimale. Infatti, fortunatamente, l'euristica è maturata tanto da assicurare che un generatore di codice ben progettato possa produrre codice diverse volte più efficiente e veloce di uno improvvisato.

Compilatori che necessitano di produrre programmi target efficienti prevedono una fase di ottimizzazione già per il codice in IR.

OTTIMIZZAZIONE DEL CODICE: dove è effettuata?

Very little scope for optimization in front end

- Computation of arithmetic expressions, simplification of logical expressions
- Any work performed here is dependent on parsing process: "syntax-directed"
- Semantic analysis begins to gather information that may help in optimization

Important optimizations take place in back end

- Adapt code to use actual registers provided, or to better exploit functional units

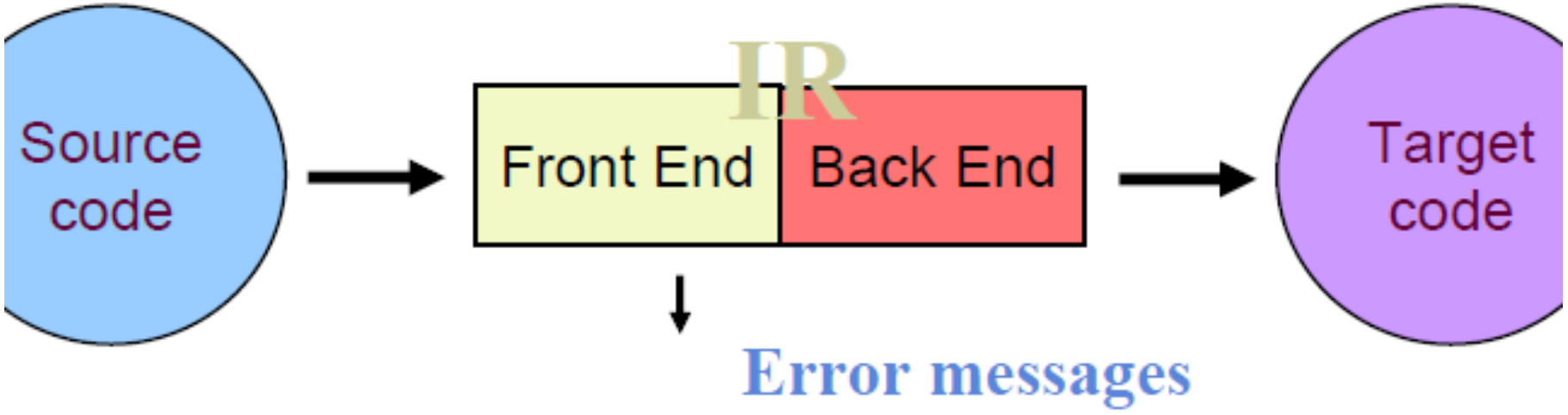
Classical back end optimizations

- **change ordering of instructions** (latency)
- **execute instructions in parallel**
- **modify data placement** (registers, cache)
- **reduce power consumption**

Ottimizzazione del codice

In aggiunta alla basilare conversione da IR a sequenza lineare d'istruzioni di macchina, un tipico generatore di codice tenta di migliorare il codice generato: usando **istruzioni del set più veloci**, riducendo il **numero delle istruzioni**, sfruttando **tutti i registri disponibili** ed evitando **computazioni ridondanti**.

Traditional Two-pass Compiler



Typically, front end is O(n) or O(n log n), while back end is NP-C (NP complete)

OTTIMIZZAZIONE DEL CODICE: ottimizzazione?

The Word "Optimization"

A clear misnomer!!

Even for simple programs, we cannot prove that a given version is optimal on a specific machine.
Let alone create a compiler to generate that optimal version ! ! !

Better:

- produce "improved" code, not "optimal" code
- can sometimes produce worst code
- range of speedup might be from 1.01 to 4 *(or more*)

OTTIMIZZAZIONE DEL CODICE: definizione

Definition

An *optimization* is a transformation that is *expected* to:
1. improve the running time of a program, ***or***
2. decrease its space requirements

Classical optimizations

reduce number of instructions ➔ reduce cost of instructions

Propagazione di costanti

X:=3; ➔ X:=3;
A:=B+X; ➔ A:=B+3;
evitando un accesso alla memoria

Eliminazione di sottoespressioni comuni

A:=B*C; T:=B*C;
D:=B*C; ⇒ A:=T;
D:=T;

Machine dependent optimization

Machines have changed since 1980

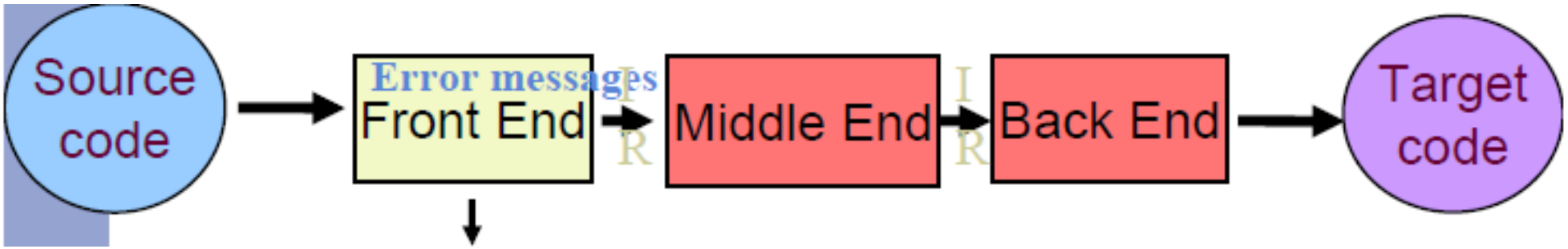
Changes in architecture implies changes in compilers

- new features present new problems
- changing costs lead to different concerns
- must re-engineer well-known solutions

- Register allocation: goal is to minimize CPU delay waiting for data (typically 60% of total execution time is actually waiting for data)
- Instruction selection
- Instruction scheduling

Compiler optimizations can significantly improve performance

Il middle end



Middle end is dedicated to code improvement

- Analyzes IR and rewrites (or transforms) IR
- Primary goal is assumed to be to reduce running time of the compiled code
- May also improve space, power consumption, …
- Improvements must provably preserve "meaning" of the code

Generatore di codice: principali task

Un generatore di codice (sia esso in IR o codice di macchina) prevede ***sempre*** tre task principali, :

- **instruction selection**, che consiste nel mapping tra programma in IR e sequenza di istruzioni che possono essere eseguite dal calcolatore target; la complessità di tale mapping è determinata da fattori quali il livello della IR, la natura dell'instruction set del calcolatore target, la qualità desiderata di codice generato.
- **register allocation and assignement**, è un requisito chiave della generazione di codice; un registro è l'unità di memoria ad accesso più veloce di una architettura, ma sfortunatamente non se ne possono avere quanti in realtà servirebbero. Perciò alcuni valori che non possono trovarsi nei registri devono risiedere in memoria e devono essere elaborati con istruzioni diverse, che occupano più spazio e sono decisamente più lente.
- **instruction ordering**, anche esso un problema NP-completo, che implica la scelta dell'ordine in cui le computazioni saranno effettute: alcune computazioni, infatti, comportano l'uso di un minor numero di registri, influenzando così l'efficienza del codice.

Instruction selection

Livello della IR

Se la IR è di alto livello, il generatore di codice può tradurre ogni statement della IR in una sequenza di istruzioni di macchina mediante predefiniti schemi di codifica (code template), che, però, richiedono una successiva fase di ottimizzazione.

Se, invece, la IR riflette alcuni dei dettagli di basso livello del calcolatore target, allora il generatore di codice può far uso di tale informazione per generare sequenze più efficienti di istruzioni di macchina.

Instruction selection (1 di 2)

Se non ci si cura dell'efficienza del programma target, l'instruction selection può essere diretta: per ciascuna quadrupla, ad esempio, si può progettare uno schema del codice target che deve essere generato per il costrutto.

Per esempio, uno statement

`x = y + z`, equivalente ad una quadrupla della forma ***`op`*** **`y,z,x`**,

dove **`x`**, **`y`** e **`z`** sono staticamente allocate,

può essere tradotto nella sequenza di istruzioni

```
LD    R0,  y          load y into register R0
ADD   R0,  R0   z     add z to R0
ST    x,   R0         store R0 into x
```

Questa strategia spesso determina operazioni "load" e "store" ridondanti.

Instruction selection (2 di 2)

Infatti la sequenza di istruzioni

```
x = y + z
w = x + v
```

sarà tradotta in

```
LD  R0, y
ADD R0, R0  z
ST  x,  R0
LD  R0, x
ADD R0, R0  v
ST  w,  R0
```

Come si può notare, la quarta istruzione è ridondante, perché carica un valore che è stato scaricato dall'istruzione immediatamente precedente. Analogamente è ridondante la terza istruzione se il valore di `x` non è usato successivamente.

Analogamente, se la macchina target disponesse di un'istruzione d'incremento `INC`, l'istruzione a 3 indirizzi `a = a + 1` potrebbe essere realizzata molto più efficientemente dalla singola istruzione `INC a` piuttosto che dalla sequenza

```
LD   R0,  a
ADD  R0,  R0  #1
ST   a,   R0
```

Strength Reduction: Replace an expensive instruction with an equivalent but cheaper alternative. For example a division may be replaced by multiplication of a reciprocal, or a list append may be replaced by cons operations.

Register allocation and assignement

L'uso dei registri comporta due ordini di problemi:

- **register allocation**, con il quale si sceglie l'insieme delle variabili che risiederanno nei registri nelle varie parti del programma;
- **register assignement**, con il quale si decide il particolare registro in cui una variabile risiederà, un problema NP-completo, difficile da risolvere anche nel caso di macchine single-register, a causa della "collisione" con il sistema operativo.

OTTIMIZZAZIONE DEL CODICE: compilatori moderni

Complexity of Modern Compilers

A commercial compiler is usually a very large and sophisticated piece of software developed over years or even decades.

A complete industrial compiler may have some millions of lines of code

E.g. Sun's Fortran compiler has ca. 4 Million Lines Of Code (MLOC)

Open64 Fortran/C/C++ has ca. 7 MLOC

No one person understands the complete system.

Optimization is . . .

Optimization is not a single process or procedure . . .

Rather it is a collection of strategies for program improvement that may be applied at various stages during compilation.

CONCLUDING

Linguaggi Formali e Compilatori ????

They are rather more complex than introductory courses suggest.