

**IA039**

**Překlad a překladače**

# Optimalizující překladač

- Základní principy práce překladače
- Mezijazyk
  - Čtveřice (obecně  $n$ -tice)
    - \* Instrukce: operátor, dva operandy, výsledek
    - \* Příklad
      - Přiřazení:  $X := Y \text{ op } Z$
    - \* Paměť: přes dočasné proměnné  $t_n$
    - \* Skoky: podmínka počítána samostatně
    - \* Skoky: na absolutní adresy

# Příklad

```
while ( j < n ) {  
k = k + j*2  
m = j*2  
j++  
}
```

# Základní překlad

```
A:: t1 := j
    t2 := n
    t3 := t1 < t2
    jmp (B) t3
    jmp (C) TRUE
B:: t4 := k
    t5 := j
    t6 := t5*2
    t7 := t4+t6
    k := t7
    t8 := j
    t9 := t8*2
    m := t9
    t10 := j
    t11 := t10+1
    j := t11
    jmp (A) TRUE
C::
```

# Základní bloky

- Program je pak reprezentován *grafem toku* (flow graph)
- Blok –část programu bez skoků
  - Jeden vstupní a jeden výstupní bod
  - Blok jako DAG (Directed Acyclic Graph)
- Optimalizace uvnitř bloků
  - Odstranění opakovaných (pod)výrazů
  - Odstranění přebytečných proměnných

# Modifikovaný překlad

```
B:: t4 := k  
    t5 := j  
    t6 := t5*2  
    m  := t6  
    t7 := t6+t4  
    k  := t7  
    t11 := t5+1  
    j  := t11  
    jmp (A) TRUE
```

# Další pojmy

- *Proměnné*
  - Definice a místa použití
- *Cykly*
- *Proces generování cílového kódu*
  - Součástí tzv. peephole optimalizace

# Optimalizovaný překlad

```
A::  t1  := j
      t2  := n
      t4  := k
      t9  := m
      t12 := t1+t1
      t3  := t1 >= t2
      jmp (B1) t3
B::  t4  := t4+t12
      t9  := t12
      t1  := t1+1
      t12 := t12+2
      t3  := t1 < t2
      jmp (B) t3
B1:: k    := t4
      m    := t9
```

# Klasické optimalizace

- Propagace kopírováním

- Příklad:

$$\begin{array}{l} X = Y \\ Z = 1. + X \end{array} \implies \begin{array}{l} X = Y \\ Z = 1. + Y \end{array}$$

- Zpracování konstant
- Odstranění mrtvého kódu

# Klasické optimalizace II

- Strength reduction

- Příklad:  $K**2 \implies K*K$

- Přejmenování proměnných

- Příklad

$x = y*z;$

$x0 = y*z;$

$q = r+x+x;$

$\implies$

$q = r+x0+x0;$

$x = a+b$

$x = a+b$

- Odstraňování společných podvýrazů (podstatné zejména pro výpočet adres prvků polí)

# Klasické optimalizace III

- Přesun invariantního kódu z cyklů
- Zjednodušení indukčních proměnných (výrazů  $s$ )
  - $A(I)$  je většinou počítáno jako:  
$$\text{address} = \text{base\_address}(A) + (I-1) * \text{sizeof\_datatype}(A)$$
což lze snadno v lineárním cyklu převést na  
*mimo cyklus:*  
$$\text{address} = \text{base\_address}(A) - \text{sizeof\_datatype}(A)$$
*v cyklu:*  
$$\text{address} = \text{address} + \text{sizeof\_datatype}(A)$$
- Přiřazení registrů proměnným

# Přenositelnost a efektivita

- Aliasing
  - Dva různé formální argumenty vs. stejné aktuální argumenty
- Typy formálních a konkrétních argumentů
- Paměť
  - Equivalence
  - Zarovnání

# Odstraňování smetí

- Podprogramy, makra
  - Inlining
- Podmíněné výrazy
  - Reorganizace složitých výrazů
  - Nadbytečné testy (`if` versus `case`)
- Podmíněné výrazy uprostřed cyklů
  - Nezávislé na cyklu
  - Závislé na cyklu
    - \* Nezávislé na iteraci
    - \* Závislé mezi iteracemi

# Odstraňování smetí II

- Redukce
  - min, max
- Skoky
- Konverze typů
- Ruční optimalizace
  - Společné podvýrazy
  - Přesun kódu
  - Zpracování polí (inteligentní překladač, C a ukazatele)

# Optimalizace cyklů

- Cíle:
  - Redukce režie
  - Zlepšení přístupu k paměti (cache)
  - Zvýšení paralelismu

# Datové závislosti

- **Flow Dependencies** (backward dependencies)

- $A(2:N) = A(1:N-1) + B(2:N)$

- **Anti-Dependencies**

- Standardně přejmenování proměnných

- Příklad

$$A(I) = B(I) * E$$

$$B(I) = A(I+2) * C$$

lze přepsat na

$$A'(1:N) = B(1:N) * E$$

$$B(1:N) = A(3:N+2) * C$$

$$A(1:N) = A'(1:N)$$

# Datové závislosti

- **Output Dependencies**

- Příklad:

$$A(I) = C(I) * 2$$

$$A(I+2) = D(I) + E$$

V cyklu je vypočteno několik hodnot konkrétní proměnné, zapsat však lze pouze tu „poslední“.

# Rozvoj cyklů (loop unrolling)

- Tělo cyklu se několikrát zkopíruje (změna velikosti iteračního kroku)
- Hlavní smysl
  - Snížení režie
    - \* Snížení počtu průchodů cyklem
  - Zvýšení paralelizace (i v rámci jednoho procesoru)
    - \* Software pipelining
- Pre- a postconditioning loops
  - Adaptace na skutečný počet průchodů

# Rozvoj cyklů II

## ■ Nevhodné cykly

- Malý počet iterací → úplný rozvoj cyklů
- „Tlusté“ (=velké) cykly: samy obsahují dostatek příležitostí k paralelizaci
- Cykly s voláním procedur: souvislost s rozvojem procedur (inlining)
- Cykly s podmíněnými výrazy: spíše starší typy procesorů
- „Rekurzivní“ cykly: cykly s vnitřními závislostmi ( $a[i] = a[i] + a[i-1] * b$ )

# Problémy s rozvojem cyklů

- Rozvoj špatným počtem iterací
- Zahlcení registrů
- Výpadky vyrovnávací paměti instrukcí (příliš velký cyklus)
- Hardwarové problémy
  - Především na multiprocесorech se sdílenou pamětí (cache koherence, přetížení sběrnice)
- Speciální případ: Rozvoj vnějších cyklů

# Asociativní transformace

- Souvisí především s paralelismem
  - Redukce
  - Skalární součin (daxpy)
  - Násobení matic
- Obracení cyklů
  - Závislé výpočty

# Optimalizace přístupů k paměti

- Optimální: nejmenší krok (práce s cache)

- FORTRAN: levý index

- C: pravý index

- Příklad:

```
DO 10 J=1,N
  DO 10 I=1,N
    A(I,J)=B(I,J)+C(I,J)*D
10 CONTINUE
```

⇒

```
DO 10 J=1,N
  DO 10 I=1,N
    A(J,I)=B(J,I)+C(J,I)*D
10 CONTINUE
```

# Optimalizace přístupů k paměti II

- Blokování

- Příklad:

```
DO 10 I=1,N
  DO 10 J=1,N
    A(J,I)=A(J,I)+B(I,J)
10 CONTINUE

DO 10 I=1,N,2
  DO 10 J=1,N,2
    A(J,I)=A(J,I)+B(I,J)
    A(J+1,I)=A(J+1,I)+B(I,J+1)
    A(J,I+1)=A(J,I+1)+B(I+1,J)
    A(J+1,I+1)=A(J+1,I+1)+B(I+1,J+1)
10 CONTINUE
```

# Optimalizace přístupů k paměti III

- Nepřímá adresace

- Příklad:

$b[i] = a[i+k] * c$ , hodnota  $k$  neznáma při překladu

$a[k[i]] += b[i] * c$

- Použití ukazatelů

- Nedostatečná paměť

- „Ruční“ zpracování

- Virtuální paměť