

Překlad a překladače

IA039

Jaro 2007

Opakování z předcházejících přednášek

Příklad
a překladače

RISC procesory

- limitovaný počet instrukcí, jednotná délka
- jednoduché adresní módy, load/store instrukce
- hodně registrů
- delay branches, brach prediction, out-of-order execution
- superskalární (MIPS – 2xFPU, 2xALU, adresace)
- superpipeline, ANDES, paralelizace

Hierarchie pamětí

- vyrovnávací paměti
- stránkování
- virtuální paměť

Optimalizující překladač

Základní principy práce překladače

- 1 překlad do mezijazyka
- 2 optimalizace
 - meziprocedurální analýza
 - eliminace mrtvého kódu, nepoužitých proměnných
 - propagace konstant
 - optimalizace cyklů
 - (paralelní optimalizace)
 - globální optimalizace
 - klasické optimalizace
 - eliminace smetí
- 3 generování kódu
 - vytížení všech jednotek, pipelining
 - optimalizace uvnitř bloku

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

Příklad
a překladače

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

```
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

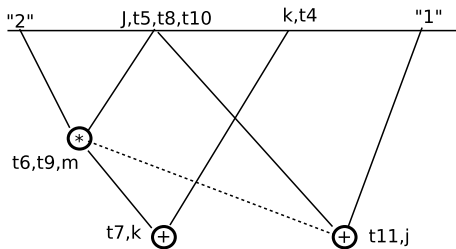
Příklad
a překladače

- 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

Flow graph

Příklad
a překladače

```
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
```



Modifikovaný překlad

Příklad
a překladače

```
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
```

```
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

Příklad
a překladače

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

Optimalizovaný překlad

Příklad
a překladače

```
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::
```

```
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
C::
```

- 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

- propagace, zjednodušování výrazů s konstantami

- Odstranění mrtvého kódu

- nedosažitelný kód
 - nepoužitý výsledek výpočtu

Klasické optimalizace II

Příklad
a překladače

- Strength reduction
 - Příklad: $K**2 \implies K*K$
- Přejmenování proměnných
 - Příklad

$x = y*z;$	\implies	$x0 = y*z;$
$q = r+x+x;$		$q = r+x0+x0;$
$x = a+b$		$x = a+b$

- Odstraňování společných podvýrazů (podstatné zejména pro výpočet adres prvků polí)
 - adresa $A(I,J) = \text{adresa}(A) + (I-1)*\text{sizeof}(\text{typ}(A)) + (J-1)*\text{sizeof}(\text{typ}(A))*\text{delkasloupce}(A)$

Klasické optimalizace III

Příklad
a překladače

- Přesun invariantního kódu z cyklů
- Zjednodušení indukčních proměnných (výrazů s)

- A(I) je většinou počítáno jako:

```
address = base_address(A) +  
(I-1)*sizeof_datatype(A)
```

což lze snadno v lineárním cyklu převést na
mimo cyklus:

```
address = base_address(A) -  
sizeof_datatype(A)
```

v cyklu:

```
address = address + sizeof_datatype(A)
```

- Přiřazení registrů proměnným

Odstraňování smetí

- Podprogramy, makra
 - Inlining
- Podmíněné výrazy
 - Reorganizace složitých výrazů
 - Nadbytečné testy (if versus case)
- Podmíněné výrazy uprostřed cyklů
 - Nezávislé na cyklu

```
DO I=1,K
  IF (N .EQ 0) THEN
    A(I)=A(I)+B(I)*C
  ELSE
    A(I)=0
  ENDIF

```

⇒

```
IF (N .EQ 0) THEN
  DO I=1,K
    A(I)=A(I)+B(I)*C
  CONTINUE
ELSE
  DO I=1,K
    A(I)=0
  CONTINUE
ENDIF

```

Odstraňování smetí II

- Redukce (vektor na skalár)

- např. min, max redukce

```
for(i=0;i<n;i++)  
    z=(a[i] > z) ? a[i] : z;
```

- rekurzivní závislost? Ano, ale:

```
for(i=0;i<n-1;i+=2)  
    z0=(a[i] > z0) ? a[i] : z0;  
    z1=(a[i] > z1) ? a[i] : z1;  
z=(z0 < z1) ? z1 : z0;
```

Odstraňování smetí III

- Skoky
- Konverze typů

```
REAL*8 A(1000)
REAL*4 B(1000)
DO I 1=1,1000
  A(I)=A(I)*B(I)
```
- Ruční optimalizace
 - Společné podvýrazy (např. obsahující volání funkce)
 - Přesun kódu
 - Zpracování polí (inteligentní překladač, C a ukazatele)

Optimalizace cyklů

Příklad
a překladače

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

- **Flow Dependencies** (backward dependencies)

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

- Příklad

$$\begin{array}{l} \text{DO } I=2, N \\ \quad A(I) = A(I-1) + B(I) \end{array} \implies \begin{array}{l} \text{DO } I=2, N, 2 \\ \quad A(I) = A(I-1) + B(I) \\ \quad A(I+1) = A(I-1) + B(I) + B(I+1) \end{array}$$

- **Anti-Dependencies**

- Standardně přejmenování proměnných

- Příklad

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

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

lze přepsat na

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

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

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

- **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“.

- problém hlavně při paralelizaci

Rozvoj cyklů (loop unrolling)

- Tělo cyklu se několikrát zkopíruje

```
DO I=1, N, 4
```

```
  A(I)=A(I)+B(I)*C
```

```
  A(I+1)=A(I+1)+B(I+1)*C
```

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

```
  A(I+3)=A(I+3)+B(I+3)*C
```

- 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

Příklad
a překladače

- 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

Příklad
a překladače

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

Rozvoj vnějších cyklů

Příklad
a překladače

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

- A(I) je v cyklu konstanta
- C(J) se prochází správně
- B(I,J) se prochází opačně!

```
DO I=1, N, 4
  DO J=1, N
    A(I+0)=A(I+0)+B(I+0, J)*C(J)
    A(I+1)=A(I+1)+B(I+1, J)*C(J)
    A(I+2)=A(I+2)+B(I+2, J)*C(J)
    A(I+3)=A(I+3)+B(I+3, J)*C(J)
```

Spojování cyklů

Příklad
a překladače

- opakované použití dat
- větší tělo cyklu
- kompilátor zvládne sám, pokud mezi cykly není jiný kód

```
for(i=0;i<n;i++)  
    a[i]=b[i]+1  
for(i=0;i<n;i++)  
    c[i]=a[i]/2  
for(i=0;i<n;i++)  
    d[i]=1/c[i+1]
```

⇒

```
a[0]=b[0]+1  
c[0]=a[0]/2  
for(i=1;i<n;i++)  
    a[i]=b[i]+1  
    c[i]=a[i]/2  
    d[i-1]=1/c[i]  
d[n]=1/c[n+1]
```


Optimalizace přístupů k paměti

- Optimální: nejmenší krok (práce s cache)
 - FORTRAN: levý index
 - C: pravý index
 - Příklad prohození cyklu:

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

Optimalizace přístupů k paměti II

Příklad
a překladače

- Někdy pomůže obrácení indexu:

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

⇒

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

- musí se udělat ručně, všude v kódu, okrajové podmínky!

Optimalizace přístupů k paměti III

- 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 IV

Příklad
a překladače

- 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

Array padding – vycpávky

Příklad
a překladače

```
COMMON // A(1024,1024)pad(129)
          B(1024,1024)pad(129)
          C(1024,1024)
DO 10 J=1,N
  DO 10 I=1,N
    A(I,J) = A(I,J)+B(I,J)*C(I,J)
  ENDDO ENDDO
```

Bez „pad(129)“

- adresa $C[1,1] = \text{adresa } B[1,1] + 1024 * 1024 * 4$
- pozice ve vyrovnávací paměti: $C[1,1] = B[1,1]$
protože při 32kB vyrovnávací paměti
 $(1024 * 1024 * 4) \bmod 32 = 0$
- přidáním vycpávky se pole posunou

Násobení matic

- A se prochází po řádku - nevyužije vyrovnávací paměť

```
DO I=1,N
  DO J=1,N
    DO K=1,N
      C[I,J]=C[I,J]+A[I,K]*B[K,J]
    ENDDO ENDDO ENDDO
```

- prohození cyklů, blok respektuje vyrovnávací paměť

```
DO IT=1,N,S
  DO KT=1,N,S
    DO JT=1,N,S
      DO I=IT,min(IT+S-1,N)
        DO K=KT,min(KT+S-1,N)
          DO J=JT,min(JT+S-1,N)
            C[I,J]=C[I,J]+A[I,K]*B[K,J]
          ENDDO ENDDO ENDDO ENDDO ENDDO ENDDO
```