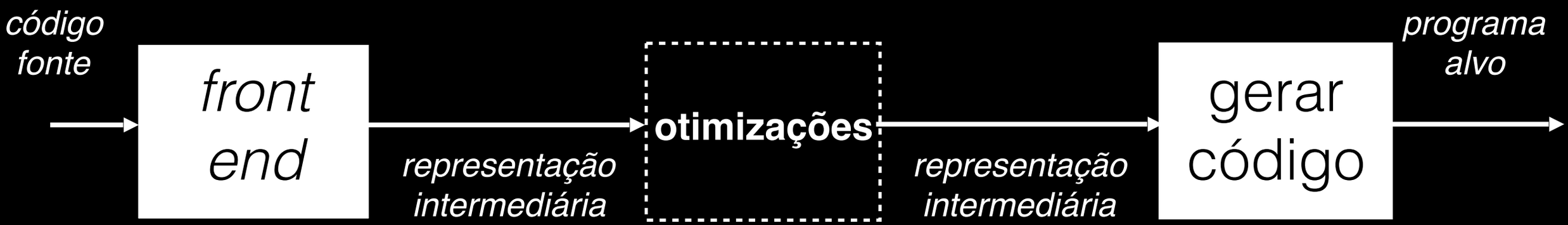
Compiladores
(IF688)

Leopoldo Teixeira
lmt@cin.ufpe.br | @leopoldomt

Geração de Código



Geração de Código

- O processo de geração se beneficia de contexto
- Podemos fazer um trabalho melhor na geração de código, se soubermos como os valores são definidos e usados
- Para isto podemos nos beneficiar de blocos básicos
    - sequências máximas de instruções de três endereços consecutivas

Gerando blocos básicos

- Dada uma sequência de instruções de três endereços, a ideia é particioná-la em uma lista de blocos básicos
- O processo para tanto, se dá pela identificação das instruções *leaders*, isto é, instruções que iniciam blocos básicos
  - a primeira instrução
  - qualquer instrução alvo de um desvio
  - qualquer instrução após um desvio

```
1)   i = 1
2)   j = 1
3)   t1 = 10 * i
4)   t2 = t1 + j
5)   t3 = 8 * t2
6)   t4 = t3 - 88
7)   a[t4] = 0.0
8)   j = j + 1
9)   if j <= 10 goto (3)
10)  i = i + 1
11)  if i <= 10 goto (2)
12)  i = 1
13)  t5 = i - 1
14)  t6 = 88 * t5
15)  a[t6] = 1.0
16)  i = i + 1
17)  if i <= 10 goto (13)
```

Usando blocos básicos

- Podemos saber quando o valor de uma variável será usado novamente
- Essencial para gerar código e alocar registradores
- A ideia é calcular *liveness* das variáveis entre blocos (como visto na aula de otimização)
- Podemos montar ou não o *flow graph*

Gerando Código

- Podemos descrever um algoritmo simples para gerar código a partir de um único bloco básico
- Considera cada instrução por vez e guarda quais valores estão em quais registradores, para evitar gerar *loads* e *stores* desnecessários

Usando registradores

- Na maioria das arquiteturas de máquina, alguns (ou todos) os operandos de uma operação devem estar em registradores
- Registradores servem bem para armazenar temporários — valores resultantes da avaliação de uma subexpressão, enquanto a maior está sendo avaliada, ou variáveis usadas apenas no bloco básico

Usando registradores

- Também são úteis para guardar valores (globais) que são computados em um bloco básico e usados em outros blocos
  - por exemplo, índices de loop
- Utilizados para auxiliar no gerenciamento de memória em tempo de execução, como é o caso de ponteiros para o topo da pilha de execução

Assumptions

- Há uma quantidade de registradores disponíveis suficiente para armazenar os valores usados dentro do bloco
- Este conjunto não inclui os registradores reservados para variáveis globais e gerenciamento da pilha
- O bloco básico já foi otimizado, eliminando subexpressões em comum, por exemplo

Assumptions

- Para cada operador há exatamente uma instrução de máquina que leva os operandos aos registradores, executa a operação e armazena resultado em registrador

- **`LD`** *reg, mem*
- **`ST`** *mem, reg*
- **`OP`** *reg, reg, reg*

Visão Geral

- A geração de código considera cada instrução e decide que *loads* são necessários para colocar os operandos em registradores
- Após gerar os *loads* o código para a operação em si é então gerado
- Se houver necessidade de armazenar o resultado em uma posição de memória, o código de *store* também é gerado

Descriptors

- Para tomar decisões sobre o que gerar, precisamos de estruturas de dados auxiliares para informar quais variáveis tem o valor atualmente em qual registrador
- Também precisamos saber se a posição de memória associada com uma dada variável tem atualmente o valor correto para esta variável
  - um novo valor para a variável pode ter sido computado em um registrador e ainda não armazenado em memória

Register Descriptors

- Para cada registrador disponível, um *register descriptor* guarda os nomes de variáveis cujo valor atual está no registrador

- Como estamos gerando código apenas para um único bloco básico, inicialmente estes *descriptors* estão vazios

- Na medida que a geração de código vai prosseguindo, vai guardar o valor de zero ou mais nomes de variáveis

Address Descriptors

- Para cada variável do programa, um *address descriptor* guarda a posição (ou posições) de memória onde o valor atual da variável pode ser encontrado
- Esta posição pode ser um registrador, um endereço na memória, uma posição na pilha...
- A informação pode ser armazenada na tabela de símbolos para aquele nome de variável

Compiladores
(IF688)

Leopoldo Teixeira
lmt@cin.ufpe.br | @leopoldomt

```
t = a - b
u = a - c
v = t + u
a = d
d = v + u
```

R1	R2	R3

a	b	c	d	t	u	v
a	b	c	d			

```
t = a - b
  LD R1, a
  LD R2, b
  SUB R2, R1, R2
u = a - c
v = t + u
a = d
d = v + u
```

R1	R2	R3
a	t	

a	b	c	d	t	u	v
a, R1	b	c	d	R2		

```
t = a - b
  LD R1, a
  LD R2, b
  SUB R2, R1, R2
u = a - c
  LD R3, c
  SUB R1, R1, R3
v = t + u
a = d
d = v + u
```

R1	R2	R3
u	t	c

a	b	c	d	t	u	v
a	b	c, R3	d	R2	R1	

```
t = a - b
  LD R1, a
  LD R2, b
  SUB R2, R1, R2
u = a - c
  LD R3, c
  SUB R1, R1, R3
v = t + u
  ADD R3, R2, R1
a = d
d = v + u
```

R1	R2	R3
u	t	v

a	b	c	d	t	u	v
a	b	c	d	R2	R1	R3

```
t = a - b
  LD R1, a
  LD R2, b
  SUB R2, R1, R2
u = a - c
  LD R3, c
  SUB R1, R1, R3
v = t + u
  ADD R3, R2, R1
a = d
  LD R2, d
d = v + u
```

R1	R2	R3
u	a,d	v

a	b	c	d	t	u	v
R2	b	c	d, R2		R1	R3

```
t = a - b
  LD R1, a
  LD R2, b
  SUB R2, R1, R2
u = a - c
  LD R3, c
  SUB R1, R1, R3
v = t + u
  ADD R3, R2, R1
a = d
  LD R2, d
d = v + u
  ADD R1, R3, R1
```

R1	R2	R3
d	a	v

a	b	c	d	t	u	v
R2	b	c	R1			R3

```
t = a - b
  LD R1, a
  LD R2, b
  SUB R2, R1, R2
u = a - c
  LD R3, c
  SUB R1, R1, R3
v = t + u
  ADD R3, R2, R1
a = d
  LD R2, d
d = v + u
  ADD R1, R3, R1
//fim do bloco
  ST a, R2
  ST d, R1
```

R1	R2	R3
d	a	v

a	b	c	d	t	u	v
a,R2	b	c	d,R1			R3

A Geração de Código

- Uma parte essencial do algoritmo é a função *getReg(I)*, que seleciona registradores para cada posição de memória associada com instrução *I*

- Esta função tem acesso aos descriptors para todas as variáveis do bloco básico em questão,

  - pode ter acesso à informações de fluxo de dados como quais variáveis estão vivas

A Geração de Código

- Vamos discutir *getReg(I)*, após apresentar como a utilizamos para gerar código
- De acordo com as nossas *assumptions*, há registradores suficientes para os dados do bloco básico
- Embora não saibamos o total de registradores disponíveis, assumimos que há registradores para suprir qualquer operação do bloco

Operações

- Para uma instrução do tipo **`x = y + z`**
  - Use *getReg(***`x=y+z`***)* para selecionar registradores para **`x`**, **`y`** e **`z`** (**R_x, R_y, R_z**)
    - Se **`y`** não está em **R_y** (*register descriptor*), emita uma instrução **`LD`** **R_y**`, y'`, onde y' é uma posição de memória para y (de acordo com o *address descriptor*)
    - Similarmente, se z não está em Rz, emita a instrução de carregamento correspondente **`LD`** **R_z**`, z'`
    - Emita a instrução **`ADD`** **R_x, R_y, R_z**

Cópia

- Para uma instrução do tipo **`x = y`**
  - Use *getReg(***`x=y`***)* para obter registradores para **`x`** e **`y`** (**R_x**, **R_y**)
    - Assumimos que esta função sempre retorna o mesmo registrador para **`x`** e **`y`**
    - Se **`y`** não está em **R_y**, emitimos instrução **`LD`** **R_y**`, y`
    - É preciso atualizar o register descriptor de **R_y** para incluir x como um dos valores encontrados lá

Finalizando o Bloco Básico

- As variáveis usadas no bloco podem acabar com seus valores apenas disponíveis em registradores
- Se for um temporário, tudo bem
- Se a variável estiver viva na saída do bloco, ou se não soubermos com certeza, precisamos assumir que o valor é necessário após saída
- Neste caso, para cada variável x cujo *address descriptor* não informa que seu valor está na memória, precisamos gerar **`ST x, R`**

Gerenciando *Descriptors*

- Na medida que são gerados *loads*, *stores* e outras instruções, precisamos atualizar os *descriptors* de acordo com regras
- Desta forma, temos como saber quais registradores estão com quais variáveis, etc

Gerenciando *Descriptors*

- Para uma instrução do tipo **`LD R, x`**
  - Altere o *register descriptor* de **`R`** para que contenha apenas **`x`**
  - Altere o *address descriptor* de **`x`** para adicionar **`R`** como localização adicional
  - Remova **`R`** do *address descriptor* de qualquer outra variável diferente de **`x`**

Gerenciando *Descriptors*

- Para uma instrução do tipo **`ST x, R`**
  - Altere o *address descriptor* de **`x`** para incluir sua própria localização na memória

Gerenciando *Descriptors*

- Para uma operação do tipo **`ADD` R_x, R_y, R_z**
  - Altere o *register descriptor* de R_x para que contenha apenas **`x`**
  - Altere o *address descriptor* de **`x`** para que R_x seja a única localização de **`x`**
  - Remova R_x do *address descriptor* de qualquer outra variável diferente de **`x`**

Gerenciando *Descriptors*

- Para operação de cópia **`x = y`**, após gerar o código para carregar **`y`** em $\mathbf{R_y}$, caso necessário, e após gerenciar *descriptors* para os *loads*:
  - Adicione **`x`** ao *register descriptor* de $\mathbf{R_y}$
  - Altere o *address descriptor* de **`x`** para que sua única localização seja $\mathbf{R_y}$

Função *getReg*

- Existem várias opções para implementar esta função para uma instrução I
- Considere uma operação como **`x = y + z`**
- Precisamos escolher um registrador para **`y`** e outro para **`z`**
- As questões são as mesmas, portanto vamos focar na escolha de um registrador pra **`y`**

Função *getReg*

- Se **`y`** já está em um registrador, escolha um dos registradores contendo **`y`** como $\mathbf{R_y}$
- Se **`y`** não está em um registrador, mas há algum vazio, escolha este como $\mathbf{R_y}$
- Se não houver registradores vazios, precisamos de qualquer forma de um registrador para **`y`**
- Seja **`R`** o registrador candidato e suponha que **`v`** é uma das variáveis que estão no *register descriptor*
- Existem algumas possibilidades…

Possibilidades

1. Se o *address descriptor* de **`v`** diz que **`v`** está em algum outro lugar além de **`R`**, ok

2. Se **`v`** for **`x`**, que é a variável sendo computada pela instrução, e **`x`** não for um dos operandos (**`x`**!= **`z`**), então ok

3. Se **`v`** não é usada mais adiante, ou seja, após instrução I, não há mais usos de **`v`**, e **`v`** está viva na saída do bloco, ela foi recomputada, então ok

4. Se não casar nenhum dos três casos acima, precisamos gerar a instrução de *store* para **`v`**, colocando uma cópia dela em sua posição de memória (*spill*)

Função *getReg*

- Agora, considere a escolha do registrador $\mathbf{R_x}$
- As questões são bem semelhantes às de $\mathbf{R_y}$, as únicas diferenças são:
  - Já que um novo valor pra **x** está sendo computado, um bom candidato é um registrador que apenas armazena **x**
  - Se **y** não é usada após instrução I, e $\mathbf{R_y}$ tem apenas **y** após o *load*, $\mathbf{R_y}$ pode ser usada como $\mathbf{R_x}$, e o mesmo se aplica para $\mathbf{R_z}$