Alocação de Registradores

Sandro Rigo
sandro@ic.unicamp.br

Introdução

- A IR e a seleção de instruções assumiram que o número de registradores era infinito
- Objetivo:
  - Atribuir registradores físicos (da máquina) para os temporários usados nas instruções
  - Se possível, atribuir a fonte e o destino de MOVES para o mesmo registrador
    - Eliminando os MOVES inúteis

Introdução

- Grafo de Interferência (IG):
  - Temos arestas entre t1 e t2 se eles não podem ocupar o mesmo registrador
  - Live ranges têm intersecção
  - Restrições da arquitetura
    - a = a + b não pode ser atribuido ao r12
- O problema se transforma em um problema de coloração de grafos

Coloração do IG

- Queremos colorir o IG com o mínimo de cores possíveis, de maneira que nenhum par de nós conectados por uma aresta tenham a mesma cor
  - Coloração de vértices
  - As cores representam os registradores
  - Considerando que a máquina tem k registradores
    - Se encontrarmos uma k-coloração para o IG
      - Essa coloração é uma alocação válida dos registradores

Coloração do IG

- E se não existir uma k-coloração?
  - Então teremos que colocar alguns dos temporários ou variáveis na memória
  - Operação conhecida como *spilling*
- Coloração de vértices é um problema NP-Completo
  - Logo, alocação de registradores também é
- Existe uma aproximação linear que traz bons resultados

Coloração por Simplificação

- Dividida em 4 fases:

1. Build:
   - Construir o IG
   - Usa a análise de longevidade
2. Simplify:
   - Heurística
   - Suponha que o grafo G tenha um nó *m* com menos de k vizinhos
   - K é o número de registradores
   - Faça G' = G – {m}
   - Se G' pode ser colorido com k cores, G também pode

Coloração por Simplificação

2. Simplify:
   - Leva a um algoritmo recursivo (pilha)
     - Repetidamente:
       - Remova nós de grau menor que K
       - Coloque na pilha
     - Cada remoção diminui o grau dos nós em G, dando oportunidades para novas remoções

Coloração por Simplificação

3. Spill:
   - Em algum momento não temos um nó com grau < k
   - A heurística falha
   - Temos que marcar algum nó para spill
   - A escolha desse nó é também uma heurística
     - Nó que reduza o grau do maior número de outros nós
     - Nó com menor custo relacionado as operações de memória

Coloração por Simplificação

4. Select:
   - Atribui as cores
   - Reconstrói o grafo G adicionando os nós na ordem determinada pela pilha
   - Quando adicionamos um nó, devemos ter uma cor para ele dado o critério de seleção usado para remover
   - Isso não vale para os nós empilhados marcados como spill
     - Se todos os vizinhos já usarem k cores, não adicionamos no grafo
     - Continua o processo

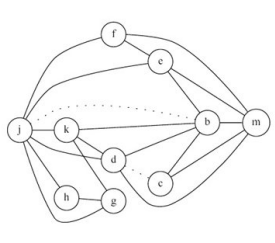
Coloração por Simplificação

5. Start Over:
   - Pode ser que o Select não consiga atribuir uma cor a algum nó
   - Reescreve o programa para pegar esse valor da memória antes de cada uso e armazená-lo de volta após o uso
   - Isso gera novos temporários
     - Com live ranges mais curtas
   - O algoritmo é repetido desde a construção do IG
   - O processo acaba quando Select tiver sucesso para todos os vértices

Exemplo

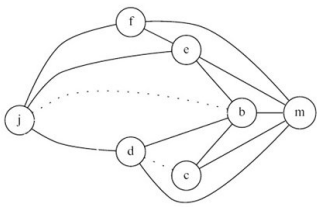
- Suponha que temos 4 registradores

```
live-in: k j
     g := mem[j+12]
     h := k - 1
     f := g * h
     e := mem[j+8]
     m := mem[j+16]
     b := mem[f]
     c := e + 8
     d := c
     k := m + 4
     j := b
live-out: d k j
```



Exemplo

- Removendo h, g , k

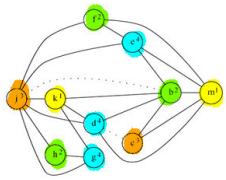


Exemplo

- Final

m	1
c	3
b	2
f	2
e	4
j	3
d	4
k	1
h	2
g	4

(a) stack (b) assignment



Coalescing

- Eliminar MOVES redundantes usando o IG
  - Se não existirem arestas entre os nós de uma instrução MOVE ela pode ser eliminada
- Os nós fonte e destino do MOVE são unidos (*coalesced*) em um só
- A aresta do novo nó é a união das arestas dos dois anteriores

Coalescing

- O efeito é sempre benéfico?
  - Qualquer instrução MOVE sem arestas no IG poderia ser eliminada
  - Pode tornar o processo de alocação mais complicado
    - Por que?
- O nó resultante é mais restritivo que os anteriores
  - Seu grau aumenta
  - Pode ser tornar >= K
- Um grafo k-colorível antes do coalescing pode ser tornar não k-colorível após uma operação de coalescing

Coalescing

- Devemos tomar cuidados
  - Executar coalescing somente quando for seguro
  - Temos duas estratégias:
- Briggs:
  - *a* e *b* podem ser unidos se o nó resultante *ab* tiver menos do que K vizinhos com grau significativo (>=K)
  - Garante que o grafo continua k-colorível. Por que?
  - Após a simplificação remover todos os nós não-significativos, sobram menos do que K vizinhos para o nó *ab*
  - Logo, ele pode ser removido

Coalescing

- George:
  - *a* e *b* podem ser unidos se para cada vizinho *t* de *a*:
    - *t* interfere com *b*
    - ou *t* tem grau insignificante (<K)
  - Por que é segura?
  - Seja S o conjunto de vizinhos insignificantes de a em G
  - Sem o coalescing, todos poderiam ser removidos, gerando um grafo G1
  - Fazendo o coalescing, todos os nós de S também poderão ser removidos, criando G2
  - G2 é um subgrafo de G1, onde o nó *ab* corresponde ao *b*
  - *G2* é no mínimo tão fácil para colorir quanto G1

Coalescing

- São estratégias conservativas
- Podem sobrar MOVES que poderiam ser removidos
- Ainda assim, é melhor do que fazer spill!

Fases da Alocação com Coalescing

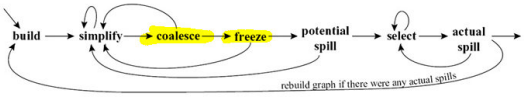
- Build:
  - Construir o IG
  - Categorizar os nós em move-related e move-unrelated
- Simplify:
  - Remover os nós não significativos (grau < K), um de cada vez
- Coalesce:
  - Faça o coalesce conservativo no grafo resultante do passo anterior
  - Com a redução dos graus, é provável que apareça mais oportunidades para o coalescing
  - Quando um nó resultante não é mais move-related ele fica disponível para a próxima simplificação

Fases da Alocação com Coalescing

- Freeze:
  - Executado quando nem o simplify nem o coalescing podem ser aplicados
  - Procura nós move-related de grau baixo.
  - Congela os moves desses nós. Eles passam a ser candidatos para simplificação
- Spill:
  - Se não houver nós de grau baixo, selecionamos um nó com grau significativo para spill
  - Coloca-se esse nó na pilha
- Select:
  - Desempilhar todos os nós e atribuir cores

Fluxo com Coalescing

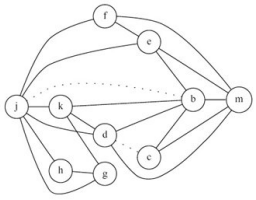
- Simplify, coalesce e spill são intercalados até que o grafo esteja vazio.



Retomando o Exemplo

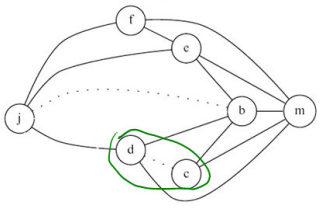
- Suponha que temos 4 registradores
- Agora somente nós não relacionados a MOVE podem ser candidatos no simplify

```
live-in: k j
        g := mem[j+12]
        h := k - 1
        f := g * h
        e := mem[j+8]
        m := mem[j+16]
        b := mem[f]
        c := e + 8
        d := c
        k := m + 4
        j := b
live-out: d k j
```



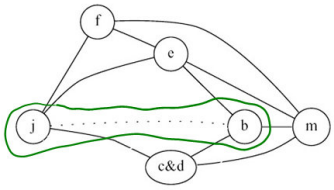
Exemplo de Coalescing

- Removendo h, g , k
- Invocando coalescing …



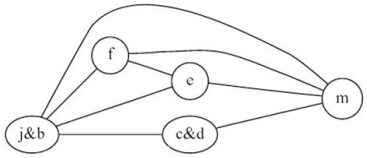
Exemplo de Coalescing

- c e d podem ser unidos
  - c&d tem 1 vizinho com grau significativo (>=K)



Exemplo de Coalescing

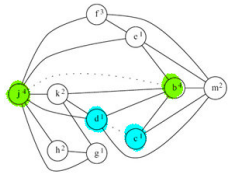
- b e j também podem ser unidos
  - j&b tem 1 vizinho com grau significativo (>=K)
- Agora simplify termina o trabalho ...



Exemplo de Coalescing

- Alocação final:

e	1
m	2
f	3
j&b	4
c&d	1
k	2
h	2
g	1
stack	coloring



Spilling com Coalescing

- Solução simples:
  - Descartar todos os coalescing feitos quando recomeçar o Build
- Mais eficiente:
  - Conservar os coalescing feitos antes do primeiro *potencial spill*
  - Descarta os subsequentes

Coalescing de Spills

- Muitos registradores => poucos spills
- Poucos registradores => vários spills
  - Aumenta o tamanho dos registros de ativação (AR)
  - Ex. Pentium: 6 registradores
- Transformações/Otimizações
  - Podem gerar mais temporários
- O frame da função pode ficar grande

Coalescing de Spills

- Instruções MOVE envolvendo valores que sofreram spill
  - a ← b implica em:
    - t ← M[b]
    - M[a] ← t
  - Caro e ainda cria mais um valor temporário
- Muitos dos valores que sofrem spill não estão vivos simultaneamente
- Podemos usar a mesma técnica que para registradores!

Coalescing de Spills

- Coloração com coalescing para os *spills*

1. Use o liveness para construir um IG para os spills
2. Enquanto houver spills sem interferência e com MOVE
   - Una esses nós (Coalescing)
3. Use simplify e select para colorir o grafo

Coalescing de Spills

3. Use simplify e select para colorir o grafo
   - Não existe spill nesta coloração
   - Simplify vai retirando o nó de menor grau até o fim
   - Select vai escolhendo a menor cor possível
     - Sem limite, pois não temos limite para o tamanho do frame
4. As cores correspondem a posições do frame da função

- Fazer **antes** da reescrita do código

Pré-coloração

- Alguns nós do IG podem ser pré-coloridos
  - Temporários associados ao FP, SP, registradores de passagem de argumentos
  - Permanentemente associados aos registradores físicos
  - Cores pré-definidas e únicas
  - Podem ser reaproveitados no *select* e *coalesce*
    - Desde que não interfiram com o outro valor
    - Ex. Um registrador de passagem de parâmetro pode servir como temporário no corpo da função

Pré-coloração

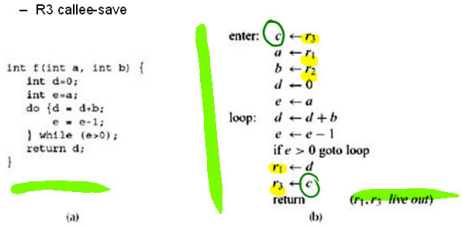
- Podem ser unidos no coalescing com outros nós não pré-coloridos
- Simplify os trata como tendo grau "infinito"
  - Não devem ir para a pilha
  - Não devem sofrer spill
- O algoritmo executa *simplify*, *select* e *spill* até sobrarem somente nós pré-coloridos

Pré-coloração

- Podem ser copiados para temporários
  - Suponha que r7 seja um callee-save register

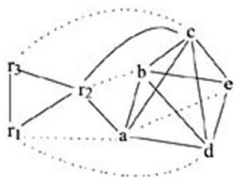
Exemplo

- Três registradores:
  - R1 e r2 caller-save
  - R3 callee-save



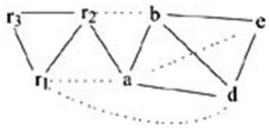
Exemplo

- IG para o programa em (b)
  - K = 3
  - Tem oportunidades para simplify e spill?



Exemplo

- Veja cálculo de prioridade para o spill na tabela do livro
  - C tem a menor prioridade



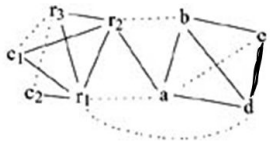
Exemplo

- Código após reescrita gerada pelo spill de c

```
enter:  c1 ← r3
        M[cloc] ← c1
        a ← r1
        b ← r2
        d ← 0
        e ← a
loop:   d ← d + b
        e ← e − 1
        if e > 0 goto loop
        r1 ← d
        c2 ← M[cloc]
        r3 ← c2
        return
```


Exemplo

- Novo IG



Exemplo

- Código alocado

```
enter:  r3 ← r3
        M[cloc] ← r3
        r1 ← r1
        r2 ← r2
        r3 ← 0
        r1 ← r1
loop:   r3 ← r3 + r2
        r1 ← r1 − 1
        if r1 > 0 goto loop
        r1 ← r3
        r3 ← M[cloc]
        r3 ← r3
        return
```


Exemplo

- Código com MOVES eliminados

```
enter:  M[cloc] ← r3
        r3 ← 0
loop:   r3 ← r3 + r2
        r1 ← r1 − 1
        if r1 > 0 goto loop
        r1 ← r3
        r3 ← M[cloc]
        return
```