Implementação de um alocador de registradores baseado em fusão

Adriano Cardoso[1]
RA 77274
ra77274@uem.br

Andrey Souto Maior[1]
RA 78788
ra78788@uem.br

Caio Tonetti[1]
RA 79604
ra79604@uem.br

Lucas Bernardes[1]
RA 80824
ra80824@uem.br

[1]Departamento de Informática
Universidade Estadual de Maringá
Maringá, PR

1 Introdução

A alocação de registradores que ocorre no *back-end*[1] de um compilador tem como objetivo designar para cada *live range* de uma variável um registrador virtual, sendo responsável por armazenar o dado corrente referente a essa variável. Quando o número de variáveis supera a quantidade de registradores, significa que não é possível atribuir um registrador físico para todos os virtuais presentes em algum bloco de instruções. Uma opção é armazenar os valores em memória -ao invés de usar registrador físico- ocorrendo o que é conhecido como *spilling*. Outra prática, é realizar *splitting*, que consiste em quebrar uma *live range* em segmentos menores diminuindo assim os conflitos de registradores físicos, porém pode ser necessário realizar *shuffle code* quando algum dos segmentos gerados é armazenado em memória -*spilled*. *Shuffle codes* passam a ser necessários, sendo realizados *loads* e/ou *stores* por ter pedaços de um live *live range* em memória.

Para a realização de forma eficiente desta etapa deve se balancear o uso de *spills* e *splits*, sendo que existem vários métodos e abordagens já conhecidos. Para a realização deste trabalho foi analisada a técnica *Fusion-based Register Allocation*(1). Proposta em 1998 por Lueh, Gross e Adl-Tabatabai(1), essa técnica utiliza uma abordagem com grafos. Seu propósito é criar um "grafo de interferência" para cada bloco de instrução, onde cada nó é a representação de uma *live range* existente naquele conjunto de instruções. Uma aresta entre dois nós representam *live ranges* que coexistem -ambas teriam que utilizar registradores físicos distintos.

Para entendermos melhor a abordagem deste trabalho, vamos primeiro apresentar um *framework* para alocação de registradores que abrange diversas técnicas e utilizá-lo para comparar com o *Fusion-based*. Esse alocador será descrito na Seção 2.

2 *Framework* de alocação de registradores

Para podermos realizar uma comparação com o *Fusion-Based*, foi proposto um alocador dividido em sete fases: *graph construction, live-range coalescing, color ordering, color assignment, graph reconstruction, spill-code insertion* e *shuffle-code insertion* (vide Figura 1). Lueh, Gross e Adl-Tabatabai(1) explicam de forma detalhada cada fase separadamente. Como o próprio nome diz, a "reconstrução de grafo" reconstrói o grafo de interferência depois de realizar *spill* de uma *live range* e recomeça a partir da fase de *coalescing*[2].

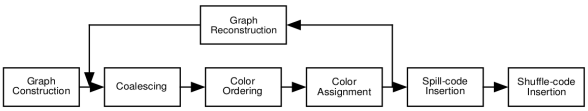

Figure 1: Estrutura do alocador de registrador

3 Alocador de registradores baseado em fusão

Alocador de registrador *Fusion-based* identifica regiões, constrói o grafo de interferência de cada região, e então realiza a junção dos grafos de todas as regiões de acordo com as arestas que ligam cada dois blocos de instrução. A fusão de grafos ocorre na fase de construção, que é dividida em três novas etapas (Figura 2): *region formation, graph simplification* e *graph fusion*.

[1]Responsável por transformar código intermediário em código específico de máquina.
[2]Processo de "juntar" duas *live ranges*

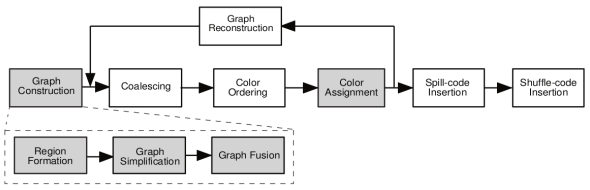

Figure 2: Estrutura do alocador de registrador baseado em fusão

A ideia principal por trás deste alocador é tentar reduzir ao máximo o número de vezes que ocorre a fase de reconstrução de grafo. Nas seções seguintes será explicado os principais aspectos do *Fusion-based*.

3.1 *Region Formation*

Fase responsável por formar "regiões". Regiões consistem em *basic blocks*[3] e as arestas que conectam esses blocos. Optou-se pelo uso de *basic blocks* devido a facilidade de utilização dos mesmos dentro da ferramenta LLVM[4] e pelo fato de que quanto menor for o tamanho de uma região menor é o custo de realizar *spill*, pois as *live ranges* podem ser divididas em uma granularidade menor (com o custo de *shuffle code*)(1).

3.2 *Graph Simplification*

O objetivo da "simplificação de grafo" é determinar quantas *live ranges* devem sofrer *spill* em cada região. Se um grafo de interferência G_R pode ser simplificado(2), então não é necessário *spill* na região R. Caso G_R não possa ser simplificado, é calculado o peso[5] P máximo de um nó do grafo e comparado com o número de registradores físicos N disponíveis para a região, sendo a diferença $(P - N)$ a quantidade de *spills* necessários. A próxima fase, *graph fusion*, determinar quais são as melhores *live ranges* para *spill*. Esta técnica é chamada de *delayed spilling*.

3.3 *Graph Fusion*

A fase de fusão entre os grafos segue a sequência de controle de arestas determinada pela região de formação onde funde os grafos de interferência por arestas. O grafo de fusão é baseado no poderoso operador de fusão que mantem a invariância que resulta no grafo de interferência simplificado. Decisões de *Live-range* são feitas pelo operador de fusão: caso fundido dois grafos *GR1* e *GR2* por um aresta E, resulta em um grafo de interferência que não pode ser simplificado, então um ou mais *Live-ranges* que ligam E sofrem *split*. Somente os *Live-range* que são ligados a E precisão ser considerados. Ao final da fase de fusão de grafo, temos um grafo de interferência simplificado; sabemos ainda quantos*live ranges* devem sofrer *spill* para cada região, mas ainda não atribuímos registradores físicos a nenhum *live range*. A invariante de de simplicidade nos permite não atribuir prematuramente cores para o grafo durante a fase de fusão.

4 Operador de fusão

Responsável por unir dois grafos de interferência em um que mantenha a condição de simplificável. A Figura 3 detalha as sub-fases da Seção 3.3.

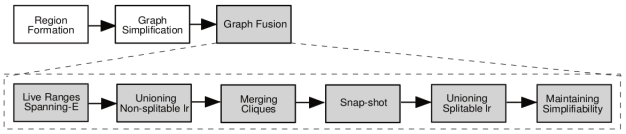

Figure 3: Estrutura do operador de fusão

4.1 *Finding Spanning-E Live Ranges*

Considerando duas regiões B_1 e B_2 e uma aresta que conecta as duas no grafo de fluxo do programa sendo compilado. Foi criado uma estrutura chamada *span_ e* que possui dois ponteiros "*from*" e "*to*". O ponteiro *from* aponta para uma *live range* de um registrador virtual x que foi, por exemplo, definido em B_1 e o ponteiro *to* aponta para uma *live range* do mesmo registrador x na região B_2.

4.2 *Unioning Nonsplitable Live Ranges*

Após identificar *span_ e*, nós juntamos as *live ranges* que são ambas marcadas para *spill*. Para isso é criado um novo nó que recebe as *live ranges* e as arestas adjacentes são convertidas para este novo nó, ocorrendo a junção de duas regiões do grafo de fluxo.

4.3 *Merging Cliques*

Quando uma região R precisa de M registradores físicos para ser colorido, e M for maior que o número real N de registradores físicos existentes, algumas *live ranges* devem sofrer *spill*. A *live range* considerada para tal foram as primeiras que aparecem no grafo de interferência.

References

1 LUEH, G.-Y.; GROSS, T.; ADL-TABATABAI, A.-R. Fusion-based register allocation. Carnegie Mellon University, 2000.

2 CHAITIN, G. J. et al. Register allocation by coloring. Research Report 8395, IBM Watson Research Center, 1981.

[3] Um *basic block* é uma sequencia de código sem entradas ou saídas de saltos (exceto pelo início e fim do próprio bloco, respectivamente.)

[4] http://llvm.org/

[5] Número de arestas ligadas a cada nó de um grafo