Estado da arte de alocação de registradores

Raoni Silva, Raul Ramalho

17 de junho de 2025

Turma de Grafos 35M34
Unidade 2

Sumário

1 Resumo

Investigação do estado da arte sobre otimização do uso de registradores com variáveis pelas técnicas de modelagem com coloração de grafos de interferência. Investigação de sua evolução histórica, mensuração dos resultados e desfecho.

2 Introdução

2.1 Contexto histórico

Registradores são componentes de memória de uso genérico de uma CPU de um computador. Sua função é de armazenar dados e intruções que serão processados de imediato. É a tecnologia de maior nível na hierárquia de memória em um computador.

Pela sua quantidade reduzida, uso intermitente e alta importância, sua ociosidade ou desperdício não é desejável. O uso ótimo destes componentes é portante prioritário. Para isso, pesquisa e técnicas de otimização foram desenvolvidas nesta área.

Entre as diversas abordagens disponíveis, uma delas adotadas fora a de coloração de grafos de interferência. Técnica que consiste em cada variável em dado programa de computador é representada por um vértice no grafo, e suas arestas representam a coexistência do tempo de vida desta variável no mesmo instante. Cada cor atribuída ao grafo corresponde ao número de registradores de propósito geral disponíveis.

Inicialmente, em 1981 nos laboratórios da IBM, o cientista Chaitin e seus colegas de pesquisa desenvolveram o primeiro algoritmo otimização do uso de registradores com coloração de grafos de interferência.

O algoritmo é especificado em quatro etapas, construção do grafo de interferência, armazenamento na memória e coloração, spilling (armazenamento na ram) e recuperação das variáveis na ram e recosntrução do grafo.

Posteriormente, em 1992, o cientista Briggs e seus colegas da Rice University, interessados no problema decidem refinar o algoritmo de Chaitin e companhia, resultando na aprimoração das heurísticas na etapa de coalescência, simplificação e spilling(derramamento).

Por fim, em 1996, pelo Bell Labs na equipe de George e Appel et al., novamente, a técnica de coloração de grafos de interferência fora otimizada mais uma vez apartir de Briggs e companhia. Agora, a técnica de coalescência admite uma nova heurística, critérios mais robustos foram atribuidas como prevenção do pior caso do algoritmo de seus antecessores. Uma nova etapa de congelamento fora adiciona antes de ocorrer o spilling. Atingindo, assim, o estado da arte.

3 Descrição do problema

O problema de alocação de registradores consiste em atribuir variáveis a um número limitado de registradores durante a compilação de um programa.

3.1 Objetivo do problema

O objetivo é estabelecer uma associação entre as variáveis do programa e os registradores disponíveis, determinando, sempre que possível, a qual registrador cada variável será atribuída.

Uma solução considerada ótima é aquela que utiliza o menor número possível de registradores para alocar todas as variáveis do programa.

Por exemplo, considere um programa com cinco variáveis. Em uma primeira tentativa de alocação, cada variável é atribuída a um registrador diferente, totalizando cinco registradores utilizados. Essa é uma solução válida. No entanto, se for possível alocar todas as cinco variáveis utilizando apenas três registradores, essa será uma solução melhor. Além disso, se o número disponível de registradores for de fato três, essa alocação representa uma solução ótima, pois atende à limitação de recursos.

Em certos casos, pode não existir uma solução válida com o número disponível de registradores, ou seja, não é possível atribuir cada variável a um registrador sem conflitos. Nesses cenários, a literatura propõe abordagens alternativas, como a técnica de *spilling*, que transfere algumas variáveis da memória rápida (registradores) para a memória principal (RAM) durante a execução.

3.2 Entradas esperadas

A entrada do problema consiste em uma descrição das variáveis do programa e suas interações, indicando quais variáveis estão ativas ao mesmo tempo e, portanto, não podem compartilhar o mesmo registrador.

Essa descrição pode assumir diferentes formas, como uma tabela de intervalos de uso de variáveis (lifetimes), uma matriz de interferência, ou qualquer outro formato que permita inferir conflitos de uso simultâneo entre variáveis.

3.3 Saídas esperadas

A saída deve indicar se existe uma atribuição válida das variáveis aos registradores disponíveis. Caso exista, a solução deve apresentar a alocação realizada, especificando a qual registrador cada variável foi atribuída.

Em caso de impossibilidade (isto é, se não for possível realizar a alocação com o número de registradores fornecido), a saída deve indicar que não há solução válida. Alternativamente, pode apresentar uma solução com *spilling*, identificando quais variáveis foram movidas para a memória.

4 Modelagem em grafos

Para a modelagem do problema, considere os vértices como representações das variáveis do programa, e as arestas indicam conflitos entre variáveis — isto é, situações em que duas variáveis estão ativas ao mesmo tempo e, portanto, não podem compartilhar o mesmo registrador. O grafo formado por essa relação é chamado de *grafo de interferência*.

Dessa forma, uma coloração adequada do grafo de interferência — atribuindo uma cor diferente para cada conjunto de variáveis que não podem ser simultâneas — corresponde a uma alocação de registradores sem conflitos.

Se a coloração do grafo puder ser feita com um número de cores menor ou igual ao número de registradores disponíveis, então é possível realizar a alocação diretamente. Caso contrário, é necessário utilizar alguma técnica de spilling para lidar com as variáveis que não puderem ser atribuidas a um registrador.

5 Estado da arte

O estado da arte foi alcançado pelo laboratório Bell Labs Innovations através do algoritmo de George e Appel et al, 1996. O algoritmo consiste no aprimoramento dos trabalho de Briggs et al, 1992. e Chaitin et al 1981.

5.1 Funcionamento do algoritmo

6 Revisão bibliográfica

A abordagem inicial da otimização do uso de registradores pelo método de coloração de grafos de interferência data do artigo publicado por Chaitin et al, 1981.

A abordagen de Chaitin consiste em uma abordagem em quatro etapas. Primeiro, é cria-se um grafo onde cada vértice representa uma variável, vértices são ligados por arestas caso a variável em questão coexista em tempo de vida com a outra. Em seguida,o algoritmo colore o grafo, caso o grau do vértice seja menor que o número de registradores, armazene-o em uma pilha. Em seguida, desempilhe-o e atribua uma cor, caso não seja possível, dado que seus vértices adjacentes já receberam uma cor no grafo original, derrame (spill) o vértice para a memória ram. Caso alguma variável esteja na ram, o processo de construção do grafo é repetido incluindo o acesso às variáveis em ram.

Em seguida, em 1986, Briggs et al. refinam o algoritmo de Chaitin, otimizando três etapas do processo.

Primeiro, otimiza a heurística de Chaitin, ao invés de empilhar o vértice com grau menor que o número de cores remove vértices com o grau maior ou igual que resultam em vértices adjacentes cujo resultado torna os demais em coloríveis.

A seleção de vértice a ser derramado é feita então por frequência de uso da variável, quanto mais usada a variável for, menor chance de ser armazenado na ram, dado a sua maior latência de acesso.

Por fim, o uso de coalescência, caso dois vértices sejam de uma mesma variável cópia que não interfiram entre si, os vértices são fundidos em um único nó.

Agora, o algoritmo de George e Appel et al. Aprimorando o trabalho de Briggs e Chaitin com regras de coalescência otimizadas de tal forma que previna que o grafo resultante não gere nós impossíveis de serem coloridos com o número de cores dados.

Parte do motivo dessa área de pesquisa estagnar por volta de 1996 ocorre pela popularização de outros métodos na otimização de registradores, como o uso de aprendizado de máquina, redes neurais de grafos e aprendizado por reforço.

7 Casos de teste

A principal comparação analisada neste trabalho é entre os algoritmos de Briggs e George-Appel. Como o algoritmo de George-Appel representa uma evolução direta sobre o de Briggs, essa comparação é especialmente relevante para evidenciar as melhorias introduzidas.

No estudo apresentado por LAL George (1996), é realizada uma comparação empírica entre os algoritmos utilizando sete programas escritos na linguagem Standard ML, compilados com o compilador SML/NJ. Esses programas foram selecionados por possuírem características variadas, tais como: diferentes tamanhos, presença ou ausência de modularização, e variáveis com durações de vida longas ou curtas. Essas variações tornam o conjunto de testes representativo de diferentes cenários encontrados em compiladores reais, conferindo robustez à comparação entre os algoritmos.

Uma limitação importante dos algoritmos originais de Briggs, identificada no estudo, é a ausência de suporte adequado para vértices pré-coloridos — situação comum em chamadas de função ou em otimizações específicas do compilador. Devido ao uso de estratégias de coalescência mais agressivas, esses algoritmos, por vezes, geravam grafos não coloríveis, impedindo a finalização do processo de alocação de registradores. Assim, a comparação entre os algoritmos foi conduzida apenas nas situações em que o algoritmo de Briggs produzia resultados viáveis, de modo a garantir uma análise justa e tecnicamente válida.

7.1 Expectativas

Visto que a melhoria do algoritmo de George-Appel é focalizada no processo de coalescência, é esperado que as melhorias se encontrem nessa parte, ou seja a quantidade de nós agrupados seja maior do que no algoritmo de Briggs. Dessa forma, reduzindo a quantidade de instruções de move ($x \leftarrow y$) no código de máquina gerado.

7.2 Expectativas

Como a principal melhoria introduzida pelo algoritmo de George-Appel está relacionada ao processo de coalescência, é natural que os ganhos de desempenho se manifestem nessa etapa. Especificamente, espera-se um aumento na quantidade de nós coalescidos, ou seja, uma maior fusão de movimentos(instruções de move, do tipo $x \leftarrow y$) entre variáveis , o que leva à geração de menos instruções de transferência no código de máquina final. Essa otimização contribui tanto para a redução do tamanho do código quanto para sua eficiência em tempo de execução.

7.3 Resultados

7.3.1 Coalescência

No primeiro conjunto de testes, a métrica avaliada foi a taxa de vértices coalescidos no grafo de interferência. O algoritmo de Briggs apresentou uma média de aproximadamente 64% de coalescência, enquanto o algoritmo de George-Appel

atingiu cerca de 86%. Esse aumento significativo confirma as expectativas iniciais e evidencia a eficácia da abordagem aprimorada. A maior taxa de coalescência implica na eliminação de mais instruções de movimento, resultando em um código de máquina mais enxuto e eficiente.

7.3.2 Tamanho do código

Como consequência direta da coalescência aprimorada, observou-se uma redução no tamanho do código assembly gerado. O algoritmo de George-Appel produziu, em média, um código 5% menor em relação ao gerado pelo algoritmo de Briggs. Essa diferença, embora modesta, pode representar ganhos importantes em ambientes com restrições de memória ou largura de banda.

7.3.3 Desempenho em tempo de execução

Por fim, as melhorias introduzidas no processo de alocação impactaram também o desempenho em tempo de execução dos programas compilados. As otimizações resultaram em uma redução média de 4,4% no tempo de execução dos casos de teste. Essa melhora reforça o argumento de que estratégias de coalescência mais eficazes não apenas reduzem o número de instruções, como também contribuem para a geração de código mais rápido.

8 Proposta de abordagem algorítmica

A proposta deste trabalho é implementar, de forma didática, o algoritmo de Iterated Register Coalescing (IRC), introduzido por George-Appel, como uma abordagem representativa e avançada para o problema de alocação de registradores baseada em coloração de grafos. A implementação visa explorar os principais mecanismos internos do algoritmo e compreender suas estratégias de otimização, com foco educacional.

A entrada para o algoritmo será um grafo de interferência já construído, representado por uma lista de adjacência. Cada nó desse grafo representa uma variável temporária, e cada aresta indica interferência entre duas variáveis — ou seja, a impossibilidade de ambas ocuparem o mesmo registrador. Essa representação abstrai etapas anteriores como análise de *liveness* e construção do grafo, as quais envolvem complexidades adicionais como o *liveness splitting*, e são consideradas fora do escopo deste projeto. Além disso, também será omitida a parte de realizar uma coloração em um grafo com nós pré-coloridos.

A implementação será realizada em Python, visando simplicidade e clareza do código. O algoritmo será organizado nas seguintes fases:

- **Simplificação**: remoção iterativa de nós com grau inferior ao número de registradores disponíveis, armazenando-os em uma pilha para posterior coloração.
- **Coalescência**: fusão segura de nós relacionados a instruções de cópia, com base em heurísticas conservadoras (como o critério de George), buscando reduzir a quantidade de instruções `move` no código final.
- **Congelamento**: paralisação de nós de cópia quando não é possível coalescê-los de forma segura, transferindo-os para a fase de simplificação.
- **Spill**: identificação e remoção de nós com alta interferência, que não podem ser alocados com os registradores disponíveis. Esses nós são marcados como *spillados* e tratados posteriormente.
- **Seleção**: desempilhamento e coloração dos nós, atribuindo registradores disponíveis. Caso não haja registrador viável para um nó, ele também é marcado como *spill*.
- **Reconstrução**: Reconstrução do grafo a partir de nós que foram considerados *spillados*, transformando-os em mais nós com menos tempos de vida e consequentemente, menos interferência com outras variáveis.

Dessa forma, será possível exemplificar o funcionamento do algoritmo de George e Appel de maneira a destacar suas otimizações utilizando heuristicas para colorir o grafo de interferência, tais como a natureza iterativa do processo de simplificação e coalescência de nós.

Espera-se que essa implementação forneça subsídios suficientes para entender as escolhas algorítmicas envolvidas em alocadores realistas de registradores, bem como para permitir análises qualitativas dos resultados obtidos, como o número de instruções de cópia eliminadas e a frequência de *spills*.

9 Conclusão

Referências

LAL George, Andrew W. Appel (jun. de 1996). “Iterated Register Coalescing”. Em: Princeton University. URL: https://dl.acm.org/doi/pdf/10.1145/229542.229546.

https://cs.gmu.edu/ white/CS640/p98-chaitin.pdf https://dl.acm.org/doi/pdf/10.1145/229542.2