Compiladores

Alocação de Registradores

Introdução

Nas notas de aula anteriores, foram explicadas formas de gerar representações intermediárias e de gerar otimizações no código durante a compilação (Aula 07 – Geração de Código Intermediário & Otimização) e a técnica de análise de longevidade para otimização de alocação de registradores (Aula 08 – Análise de Longevidade).

A continuação do conteúdo versa sobre as estratégias de alocação de registradores. O objetivo destas notas de aula é definir e apresentar a etapa de alocação de registradores e identificar formas de implementação de um alocador de registradores.

Registradores

Uma das etapas da compilação, dentro da síntese, é a ***alocação de registradores***. Mas o que é um registrador?

A maioria das máquinas possuem um conjunto de registradores, espaços de memória dedicados, dentro da Unidade Central de Processamento, que podem ser acessados com rapidez e realizar cálculos. Esses registradores estão presentes na hierarquia de memória como o recurso de memória mais ao topo, como pode ser visto na Figura 1. O registrador é a memória mais cara, consequentemente a mais escassa.

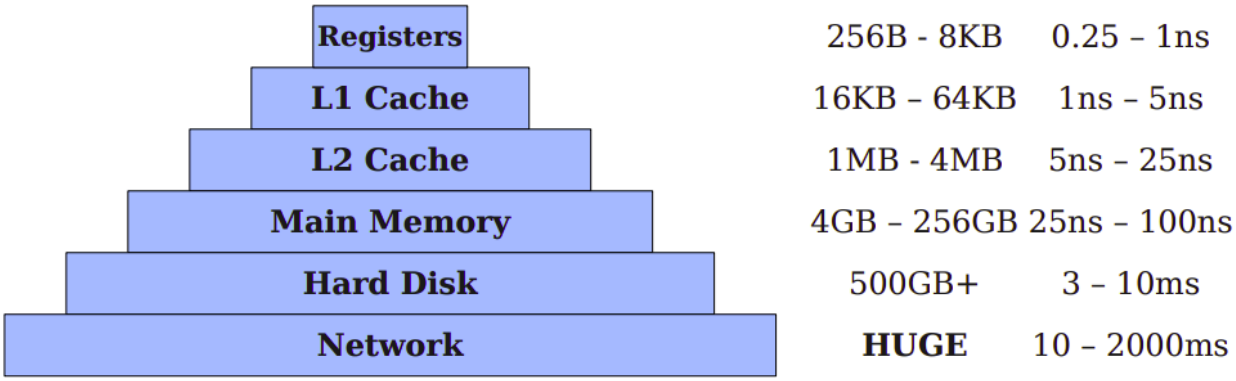

Figura 1. Hierarquia de Memória. Fonte: Schwarz, s.d.

Alocação de Registradores

Ao gerar a(s) representação(ões) intermediária(s), possivelmente otimizadas, e buscar e selecionar as instruções de máquinas que equivalem ao código escrito em alto nível pelo programador, o compilador não leva em consideração a limitação de registradores da arquitetura. A Figura 2 traz algumas máquinas e as respectivas quantidades de registradores de propósito geral. Embora a figura não esteja atualizada, é possível perceber o quão escasso é esse recurso.

A alocação de registradores é um processo que atribui variáveis a registradores e gerencia a transferência de dados entrando e saindo dos registradores. Nesse processo, dado que a quantidade de registradores é finita e pequena na maioria das arquiteturas, ocorre a decisão de quais variáveis ficam em registradores e quais ficam em memória. Essa decisão é crítica, pois o acesso à memória é muito mais lento. Assim, um bom código de alocação de registradores deve fazer bom uso dos registradores e evitar ao máximo a necessidade de acessos à memória, reduzindo assim o tempo de execução.

Considerando o número finito e pequeno de registradores, é comum um registrador armazenar valores de variáveis diferentes em determinados momentos, o que significa dizer que esses registradores costumam ser reusáveis. No entanto, se duas variáveis estiverem vivas ao mesmo tempo, não é possível que elas compartilhem o mesmo registrador. Desta forma, precisam estar alocadas em registradores diferentes ou, em caso de impossibilidade, precisam

ser armazenadas na memória RAM (*Random Access Memory*), o que torna a execução mais lenta. Outra possibilidade é a arquitetura prever em seu projeto o uso de *cache* de registradores. Neste caso, uma estratégia comum é armazenar em *cache* os registradores mais acessados, fazendo com que um mesmo registrador possa ser fonte e destino em uma operação de `mov`[1] com eficiência.

Machine	Number of general-purpose registers	Architectural style	Year
EDSAC	1	Accumulator	1949
IBM 701	1	Accumulator	1953
CDC 6600	8	Load-store	1963
IBM 360	16	Register-memory	1964
DEC PDP-8	1	Accumulator	1965
DEC PDP-11	8	Register-memory	1970
Intel 8008	1	Accumulator	1972
Motorola 6800	2	Accumulator	1974
DEC VAX	16	Register-memory, memory-memory	1977
Intel 8086	1	Extended accumulator	1978
Motorola 68000	16	Register-memory	1980
Intel 80386	8	Register-memory	1985
MIPS	32	Load-store	1985
HP PA-RISC	32	Load-store	1986
SPARC	32	Load-store	1987
PowerPC	32	Load-store	1992
DEC Alpha	32	Load-store	1992

Figura 2. Quantidade de registradores de propósito geral, estilo de arquitetura e ano de fabricação de algumas máquinas. Fonte: Centoducatte, 2007.

Segundo Rigo (2009), a alocação de registradores tem como objetivos:

- Atribuir registradores físicos (da máquina) para os temporários usados nas instruções; e
- Se possível, atribuir a fonte e o destino de `MOVES` para o mesmo registrador, eliminando operações de `mov` inúteis.

Segundo Torczon & Cooper (2011), a alocação de registradores apresenta grande

[1] Uma instrução `mov` realiza a cópia de dados de um operando (registrador, memória ou valor constante), para um local referido por outro operando (registrador ou memória).

complexidade computacional. O problema normalmente é modelado com o algoritmo de coloração de grafos, que é NP-Completo[2]. O problema de coloração de grafos pode ser reduzido ao problema de alocação de registradores mostrando que, para um grafo arbitrário, um programa pode ser construído de tal forma que a alocação de registradores para o programa, com registradores representando nós e registradores de máquina representando cores disponíveis. seria uma coloração para o gráfico original.

Implementação do Alocador de Registradores

De acordo com Schwarz (s.d.), há pelo menos três algoritmos úteis para a implementação de um alocador de registradores:

1) Alocação ingênua
2) Varredura linear
3) Coloração de grafos

Alocação ingênua

Este é o cenário mais simples. Na alocação ingênua, a ideia é armazenar todos os valores na memória principal, carregando-os apenas quando forem necessários.

- Gera instruções de carregamento (`LOAD`) para buscar os valores da memória principal para os registradores.
- Gera instruções de armazenamento (`STORE`) para armazenar o resultado de volta na memória principal.
- Gera códigos para realizar as operações entre os registradores (por exemplo, a soma – `ADD`).

O código abaixo serve de exemplo para elucidar a ideia de um alocador de registradores ingênuo:

```
a = b + c;
```

[2] Um problema é chamado NP (polinômio não determinístico) se sua solução pode ser adivinhada e verificada em tempo polinomial; não determinístico significa que nenhuma regra particular é seguida para fazer a suposição. Se um problema é NP e todos os outros problemas NP são redutíveis em tempo polinomial a ele, o problema é NP-completo (BRITANNICA, 2022).

```
d = a;
c = a + d;
```

O código faz três atribuições, às variáveis `a`, `d`, `b`, sendo duas destas atribuições compostas por expressões matemáticas de soma (`a=b+c` e `c=a+d`) e uma simples cópia de valor (`d=a`). Suponha que os valores são de ponto flutuante, representados com 32 *bits*, e que eles estão armazenados na memória na seguinte ordem:

Variável	**Posição memória**
`a`	0
`b`	4
`c`	8
`d`	12

Suponha também que expressões na forma `$tn` são representações de registradores. A letra `t` indica aqui que eles são temporários. Assim, em uma arquitetura hipotética, as operações ficariam da seguinte maneira:

```
a = b + c;
LOAD $t0, mem(4)
LOAD $t1, mem(8)
ADD $t2, $t0, $t1
STORE $t2, mem(0)

d = a;
LOAD $t0, mem(0)
STORE $t0, mem(12)

c = a + d;
LOAD $t0, mem(0)
LOAD $t1, mem(12)
ADD $t2, $t0, $t1
STORE $t2, mem(8)
```

Conjunto de instruções completo:

```
LOAD $t0, mem(4)
LOAD $t1, mem(8)
ADD $t2, $t0, $t1
STORE $t2, mem(0)
LOAD $t0, mem(0)
STORE $t0, mem(12)
LOAD $t0, mem(0)
LOAD $t1, mem(12)
ADD $t2, $t0, $t1
STORE $t2, mem(8)
```

Nesta alocação, percebe-se que há operações desnecessárias, tornando o alocador ineficiente. A quarta e a quinta operação, por exemplo, poderiam ser evitadas, pois o valor de `a` já está em um registrador e poderia ser atribuído diretamente a `d`. As vantagens deste alocador são a simplicidade e a despreocupação com a falta de registradores. No entanto, a grande quantidade de acessos à memória o tornam inviável para uso em um compilador real.

Varredura Linear

Um algoritmo de Varredura Linear (*Linear Scan Algorithm*) direciona a alocação global de candidatos a registradores para registradores com base em uma varredura linear simples sobre o programa que está sendo compilado. Essa abordagem de alocação de registradores faz sentido para sistemas, como aqueles para compilação dinâmica, onde a velocidade de compilação é importante (TRAUB; HOLLOWAY; SMITH, 1998).

Segundo Traub, Holloway & Smith (1998), os métodos de varredura linear de alocação de registradores são rápidos e eficazes. Eles podem permitir a otimização de grandes programas e são apropriados para a geração de código em tempo de execução. Eles evitam o risco de degradação do desempenho em tempo de compilação que os métodos de coloração de grafos sofrem em determinadas entradas do programa. No entanto, a maioria dos compiladores comerciais e de otimização de pesquisa dependem de uma abordagem de coloração de grafos para a alocação global de registradores.

Neste algoritmo, o código é percorrido linearmente para encontrar variáveis e determinar seus intervalos de vida (*live ranges*). Os *live ranges* são determinantes para alocar o mínimo possível de registradores e reutilizá-los quando possível. No entanto, um problema disso é não identificar os períodos em que a variável está viva mas não está ativa.

Se em algum momento não for possível alocar todas as variáveis nos registradores disponíveis (ou seja, quantidade_de_variáveis_necessárias > quantidade_de_registradores), alguma(s) variável(is) – em geral, a(s) menos utilizada(s) – precisará(ão) ser alocada(s) na memória RAM. Isso é chamado *derramamento* ou *spill*.

Coloração de grafos

Esta abordagem modela o problema de alocação de registradores como o problema de ***coloração de grafos***.

Os vértices do grafo representam o *live range* de cada variável, e as arestas representam a ligação entre dois *live ranges*. Portanto, se há uma aresta entre dois vértices, significa que as variáveis representadas por estes *live ranges* não podem compartilhar o mesmo registrador, ou seja, em outras palavras, elas existem ao mesmo tempo em algum momento de suas vidas.

As cores são atribuídas aos vértices de forma que todos os vértices adjacentes, ou seja, que tenham uma ligação direta, não podem ter a mesma cor. Ao aplicar este algoritmo, o número mínimo de cores necessárias para colorir o grafo é o número mínimo de registros necessários. Portanto, a alocação mínima será realizada por *k* registradores se o grafo for *k*-colorível.

Segundo Rigo (2009), é necessário colocar alguns dos temporários ou variáveis na memória (operação *spilling*), se não houver uma *k*-coloração.

Conclusão

A alocação de registradores é um processo essencial e complexo no desenvolvimento de compiladores, com um impacto significativo no desempenho do código gerado. Como discutido, existem várias estratégias para implementar um alocador de registradores, cada uma com suas

vantagens e desvantagens. A alocação ingênua, apesar de simples, é ineficiente devido ao excesso de operações de carga e armazenamento, o que resulta em um tempo de execução maior. A varredura linear oferece uma abordagem rápida e eficaz para a alocação de registradores, especialmente útil em sistemas que exigem uma compilação dinâmica e rápida. No entanto, ainda pode necessitar de soluções como o *spill* para variáveis que não cabem em registradores disponíveis. A coloração de grafos, embora computacionalmente mais complexa, é uma das abordagens mais robustas para a alocação de registradores. Ao modelar o problema como um grafo, é possível determinar a alocação mínima de registradores necessários, garantindo que variáveis vivas ao mesmo tempo não compartilhem o mesmo registrador. No entanto, essa técnica também pode exigir a transferência de algumas variáveis para a memória quando a coloração do grafo não é possível com o número de registradores disponíveis.

Exercícios

1) No contexto da alocação de registradores, o que é *Coalescing*?
2) Pesquise e elabore um exemplo de código demonstrando e explicando passo a passo o funcionamento da coloração de grafos.

Referências

BRITANNICA. NP-complete problem. Disponível em: https://www.britannica.com/science/mathematics. Acesso em: 02 jun. 2024.

CENTODUCATTE, P. C. Organização de Computadores: Teoria e Prática. 2007. Disponível em: https://www.ic.unicamp.br/~ducatte/mc542/Slides/mc542_A_01_2s07.pdf. Acesso em: 02 jun. 2024.

RIGO, S. Alocação de Registradores. s.d. Disponível em: https://ic.unicamp.br/~sandro/cursos/mc910/slides/cap11-regalloc.pdf. Acesso em: 02 jun. 2024.

SCHWARZ, K. Register Allocation. Disponível em: https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/lectures/17/Slides17.pdf. Acesso em: 02 jun. 2024.

TORCZON, L.; COOPER, K. Engineering A Compiler. 2nd. ed. San Francisco, CA, USA:

Morgan Kaufmann Publishers Inc., 2011. ISBN 012088478X.

TRAUB, Omri; HOLLOWAY, Glenn; SMITH, Michael D. Quality and speed in linear-scan register allocation. ACM SIGPLAN Notices, v. 33, n. 5, p. 142-151, 1998. Disponível em: https://dash.harvard.edu/bitstream/handle/1/34325454/tr-21-97.pdf?sequence=1&isAllowed=y. Acesso em: 02 jun. 2024.