Segunda Lista de Exercícios de Arquiteturas/PAD

Felipe Menino Carlos

01/05/2020

Objetivo

O principal objetivo desta lista de exercícios é fixar conceitos vistos em sala de aula sobre hierarquias de memória, em particular com relação a memórias cache.

Exercícios

1. Sabe-se que um programa, ao ser executado num certo sistema, apresentou a seguinte sequência de endereços hexadecimais nos seus acessos à memória:

   022, 014, 035, 105, 034, 034, 035, 100, 053, 035, 014, 047, 008, 105, 014, 100, 035, 003, 008, 020

Supondo que o sistema contenha uma cache de 16 entradas, utilizando mapeamento direto, e que a cache estava inicialmente vazia, determine o resultado (isto é, M: miss ou H: hit) para cada acesso da sequência mostrada. Determine também a taxa total de acerto (hit-ratio) na cache.

`Sugestão`: monte um diagrama com o estado da cache após cada acesso

Obs.: Para a realização deste exercício, foi assumido que cada bloco de memória contém 1 byte, e que as operações de movimentação dos blocos entre memória principal e memória Cache são feitos byte a byte.

Como visto durante as aulas, a memória Cache desempenha um papel muito importante para o desempenho de computadores, uma vez que esta é resposável por tornar disponível de forma rápida ao processador dados que estão sendo utilizados no processamento, evitando a necessidade de recuperar dados na memória principal, que por vezes pode ser um impeditivo para o desempenho.

Dentre as principais características da memória Cache estão sua alta velocidade e sua pequena quantidade de armazenamento, além de sua forma de organização em níveis hierárquicos. Por contar com uma pequena quantidade de memória disponível, faz-se necessário a utilização de uma função de mapeamento (Stallings 2010), que mapeia os dados entre a memória principal e a memória Cache. Este exercício utiliza de umas das técnicas de mapeamento, o `Mapeamento direto`, onde cada bloco da memória principal fica vinculado a somente um bloco da memória Cache.

A Tabela abaixo apresenta as mudanças da memória Cache frente a cada um dos acessos de memória realizados. Para o entendimento da atividade, uma representação gráfica [1] também foi criada.

N° do acesso	Endereço Hexadecimal	Endereço Binário	Hit ou Miss	Bloco de cache atribuído
1	0x22	100010	Miss	0010
2	0x14	10100	Miss	0100
3	0x035	110101	Miss	0101
4	0x105	100000101	Miss	0101
5	0x034	110100	Miss	0100
6	0x034	110100	Hit	0100
7	0x035	110101	Miss	0101
8	0x100	100000000	Miss	0000
9	0x053	1010011	Miss	0011
10	0x035	110101	Hit	0101
11	0x014	10100	Miss	0100
12	0x047	1000111	Miss	0111
13	0x008	1000	Miss	1000
14	0x105	100000101	Miss	0101
15	0x014	10100	Hit	0100
16	0x100	100000000	Hit	0000
17	0x035	110101	Miss	0101
18	0x003	0011	Miss	0011
19	0x008	1000	Hit	1000
20	0x020	100000	Miss	0000

A Tabela acima apresenta cada um dos acessos e as respectivas mudanças realizadas dentro da memória Cache. Com isto, ao final de todos os acessos tem-se as seguintes memórias válidas.

Endereço de memória Cache	0010	0100	0101	0000	0011	0111	1000
Flag	000	000	000	000	000	000	000
Bit de validação	1	1	1	1	1	1	1

A determinação do *hit-ratio* é defina da seguinte forma

$$hitRatio = \frac{hit_{total}}{acesso_{total}}$$

Com isto, o *hitRatio* deste programa é:

$$hitRatio = \frac{5}{20} = 0.25 \; ou \; 25\%$$

[1] Disponível em: https://bit.ly/2KSMoV6

2. Considere o programa `prog.f` e a subrotina continda no arquivo `mysecond.c`, disponíveis na home-page do curso. Pede-se o seguinte:

- a. Compile este programa em qualquer sistema (por exemplo com gcc e gfortran), execute-o e informe os resultados, incluindo o tempo de execução obtido.

Soma	0.0000000000000000
Tempo	0.5097181797027587 9

- b. Altere o segundo loop duplo do programa, invertendo tais loops (isto é, inverta os loops i e j). Compile e execute esta versão modificada, e informe os resultados da execução.

Soma	0.0000000000000000
Tempo	0.16058301925659180

- c. Qual das versões acima do programa (isto é, com loops i/j ou j/i) roda mais rapidamente? Explique a razão para que isto ocorra.

A segunda versão roda mais rápido. Isso ocorre por conta da maneira com que o Fortran faz o armazenamento de arrays multidimensionais na memória. Wadleigh and Crawford (2000) especificam que, diferente de C que armazena os arrays multidimensionais linhas principais, Fortran faz este armazenamento através de colunas principais, assim, ao alterar o loop, o programa passou a utilizar melhor tal característica e evitar possíveis Cache Missing e recuperações desnecessárias de memória.

Referências

Stallings, William. 2010. *Arquitetura E Organização de Computadores*. 8° Edição. São Paulo: Pearson.

Wadleigh, Kevin R, and Isom L Crawford. 2000. *Software Optimization for High Performance Computing*. 1° Edição. Hawlett-Packard Professional Books.