ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES

Luiza Batista Laquini – 2019107786

8. Arquitetura de Memória

Objetivos

Após completar as atividades deste laboratório, você irá:

- Entender o funcionamento do cache e de hierarquia de memória;
- A influência da organização da cache no desempenho de um programa.

Introdução

Caches tiram vantagens das estatísticas de acesso para melhorar o desempenho de tempo de acesso. Programas não acessam RAM aleatoriamente. Os programas fazem acessos à posições de memória exibindo localidade temporal e espacial das palavras acessadas e as caches exploram esse comportamento para melhorar o desempenho da execução destes programas. A localidade temporal está baseada no princípio de que quando uma palavra é acessada, é provável que ela seja acessada novamente nos ciclos seguintes. A localidade espacial diz que, se um programa acessa uma palavra de memória, é provável que ele acesse as palavras vizinhas nos ciclos seguintes.

Para as etapas a seguir, você deverá utilizar o programa `Vector_With_Random_Numbers.asm`, que cria um vetor de N=10 posições no segmento de dados e, em seguida, preenche o vetor com valores aleatórios (no intervalo entre 0 e 10) ou com valores digitados pelo usuário. Em seguida, o programa imprime os valores de cada posição deste mesmo vetor.

Tarefa 1

Após copiar e executar o programa com os valores dados, observe (e entenda) as funções dos valores `array`, `size` e `max` no programa.

Q1: Altere os valores de `array`, `size` e `max` para que o programa, ao ser executado, crie um vetor de 100 posições, preenchido com valores aleatórios entre 0 e 9999. Quais são as chamadas de serviço responsáveis pela geração dos valores inteiros aleatórios? Qual foi a saída gerada?

R: Os valores aleatórios são gerados por 3 chamadas de serviço. São elas:

- syscall 30 (system time);

- syscall 40: Define a semente do gerador de número pseudoaleatório Java subjacente correspondente (java.util.Random));
- syscall 42: a0 contém um valor int pseudoaleatório uniformemente distribuído no intervalo 0 <= [int] <[limite superior], extraído da sequência deste gerador de número aleatório.

As linhas correspondentes no programa são:

1. addi $v0, $zero, 30 # Syscall 30: System Time syscall
2. addi $v0, $zero, 40 # Syscall 40: Random seed
3. addi $v0, $zero, 42 # Syscall 42: Random int range

Saída gerada:

3985 3985 3985 3985 3985 3985 3985 3985 3985 3985 3985 5418 5418 5418 5418 5418 5418 5418 5418 5418 5418 5418 3025 3025 3025 3025 3025 3025 3025 3025 3025 3025 3025 3025 767 767 767 767 767 767 767 767 767 1993 1993 1993 1993 1993 1993 1993 1993 1993 1993 1993 2070 2070 2070 2070 2070 2070 2070 2070 2070 2070 271 271 271 271 271 271 271 271 271 271 271 271 271 271 2644 2644 2644 2644 2644 2644 2644 2644 2644 2644 2644 2644 2644 2644 2644 2644 845 845 845

Tarefa 2

Q2: Altere novamente os valores de `array`, `size` e `max` para que o programa, ao ser executado, crie um vetor de 33 posições, preenchido com valores aleatórios entre 0 e 99. Abra o componente "Tools→Memory Reference Visualizer" no MARS e clique em "Connect to MIPS". Em seguida, abra o componente "Tools→ Data Cache Simulator". Observe os parâmetros para organização da cache na parte superior da janela deste último componente.

Configure uma pequena cache de 512 bytes, com 32 blocos de 4 palavras, mapeamento direto e troca de palavras pela política LRU. Anote o número de acessos, o de acertos (*hits*) e falhas (*misses*) ocorrido. Compare esses resultados com o número de `lw` e `sw` do código executado. Explique o que causou cada uma das falhas de cache na execução (não se desespere porque o número de falhas é muito pequeno!). Marque a opção "*Runtime log*" *Enabled* para facilitar sua resposta.

R: Ocorreram um total de 889 acessos à memória (871 hits e 18 misses), configurando uma taxa de 98% de acerto. Os poucos erros podem ser justificados por tentativa de acesso de dados que ainda não

estavam inseridos na memória ou escrita em locais indevidos, sendo esses dois fenômenos gerados

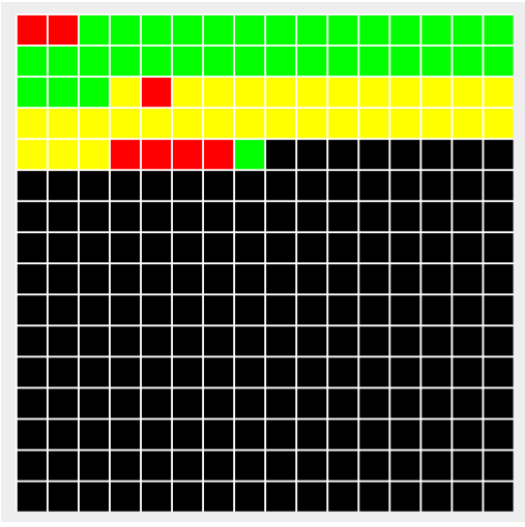
pelas instruções lw e sw, pois ambas fazem acesso à memória.

Q3: Altere a organização da cache de acordo com os parâmetros indicados e preencha as linhas da tabela a seguir.

Número de blocos	Tamanho do bloco da cache	Política de troca de bloco	Política de mapeamento	Número de caminhos	Número de acessos	Número de hits	Número de misses	Taxa média de acerto
32	4	Random	Mapeamento direto	1	889	871	18	98%
32	4	LRU	Totalmente associativo	32	889	871	18	98%
32	4	LRU	Associativo por conjunto	4	889	871	18	98%
64	4	LRU	Mapeamento direto	1	889	871	18	98%
64	8	LRU	Mapeamento direto	1	889	880	9	99%
128	4	LRU	Mapeamento direto	1	889	871	18	98%
128	4	Random	Mapeamento direto	1	889	871	18	98%

Para facilitar o entendimento do que acontece durante a execução do programa, reduzindo a velocidade de execução (instruções por segundo) clicando no componente abaixo da interface do MARS:

Q4: Usando o componente "Tools→Memory Reference Visualizer", mostre qual é o padrão final dos acessos à memória de dados e à memória de código para a execução do código com os mesmos parâmetros da Q2. Explique os padrões de cores gerados, usando como base (a) os acessos à memória de dados (instruções `lw` e `sw`) e à de instruções (PC+4) e (b) as cores associadas às células de memória (o parâmetro "*Counter Value*" associa o vermelho aos endereços com 10 ou mais acessos e o azul, a um único acesso durante a execução do programa).

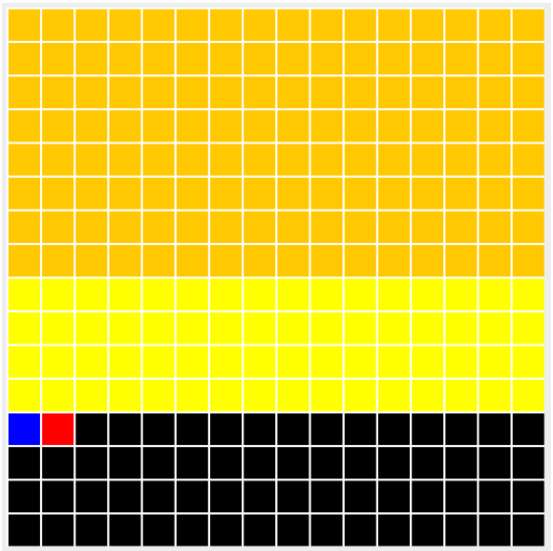


R: Os quadrados vermelhos iniciais são causados pela quebra de linha e pelo caractere ";", pois ambos são impressos com certa frequência. A sequência de quadrados verdes representa os valores que são gerados aleatoriamente, não sendo usados até a impressão. A primeira caixa amarela é o tamanho do

vetor, sendo utilizado uma vez por loop. O próximo vermelho é o limite máximo de aleatórios, que é utilizado sempre que um novo aleatório é gerado. A sequência de amarelos é a solicitação de entrada, em forma de string. Os quatro vermelhos representam a string “Input Value#”, que é impressa uma vez para cada componente do vetor. Por último, o ponto verde é a string “V=”, que só é carregada na memória e impressa uma vez.

Q5: Repita a Q3/Q4 para o programa `Mult_Matriz_Inteiros 8x8.asm`. Reduza a velocidade de execução e observe os acessos aos elementos das linhas e colunas de cada uma das matrizes usadas na multiplicação.

R:



Número de blocos	Tamanho do bloco da cache	Política de troca de bloco	Política de mapeamento	Número de caminhos	Número de acessos	Número de hits	Número de misses	Taxa média de acerto
32	4	Random	Mapeamento direto	1	1361	1176	185	86%
32	4	LRU	Totalmente associativo	32	1361	1298	63	95%
32	4	LRU	Associativo por conjunto	4	1361	1176	185	86%
64	4	LRU	Mapeamento direto	1	1361	1312	49	96%
64	8	LRU	Mapeamento direto	1	1361	1336	25	98%
128	4	LRU	Mapeamento direto	1	1361	1312	49	96%
128	4	Random	Mapeamento direto	1	1361	1312	49	96%

Os primeiros blocos laranjas são as células da matriz, utilizadas para realizar a multiplicação. As células amarelas são o resultado da multiplicação, utilizados quando são gravados na memória e impressos. O bloco azul é a quebra de linha inicial da disposição da matriz, e o último vermelho são as tabulações que espaçam todas as células da matriz.