Universidade Federal de Pelotas – UFPel

Arquitetura e Organização de Computadores II

Simulador de Cache

Grupo: Mateus Brugnaroto,
Vinicius Geraldo.

Professor: Marcelo Porto.

Pelotas, RS – 16 de novembro de 2018.

Introdução:

O trabalho realizado tem o intuito de criar um simulador de memória cache implementado em linguagem de programação C. Partindo dessa ideia, objetivo final tem como verificar a quantidade de miss e hit na cache a partir de uma determinada configuração e de um fecho de dados.

O trabalho foi divido em “headers”. Cada header tem uma função especifica.

Headers:

1 - Simulador.c: contém a chamada de todas as funções das headers para a simulação. Faz a chamada para a captação dos dados do usuário sobre as dimensões da cache, os cálculos da tag, indice e offset (faz as operaões necessárias de conversão), a instanciação da cache dos dados solicitados pelo processador.

2 - Cache.c: faz a parte de criar a cache, inicializando com valores que sejam diferentes para qualquer dado processado (Exemplo: índice “-1”), impressão dos dados contidos na cache, inserção/busca dos dados no cache e toda a verificação de miss e hit na cache. Na inserção/busca dos dados no cache, o código é “divido” em duas partes: a primeira é quando é Mapeamento Direto (associatividade igual a 1) e a segunda parte para quando for Mapemanto Associativo por Conjuntos e Mapeamento Totalmente Associativo (associatividade maior que 1).

3 - LeituraArquivo.c: lê de um arquivo todos os dados solicitados pelo processador.

4 - LeituraInformação.c: lê as configurações da cache solicitadas pelas usuário.

5 - ConversorBinario.c: converte o dado processado para seu valor em inteiro.

6 - Tag.c: seleciona os bits da tag do dado.

7 - Indice.c: seleciona os bits do índice do dado.

8 - Cálculo.c: calcula o endereço do dado na cache.

9 - CálculoBits.c: calcula quantos bits o offset ou índice utilizam.

10 - PalavrasBloco.c: calcula quantas palavras tem cada bloco da cache.

11 - TabelaDeDados.c: mostra na tela os dados de miss e hit.

12 - Random.c: função para calcular uma posição aleatória na cache.

O código utiliza de duas políticas de substituição, a random (a partir do header “Random.c” faz o cálculo para obter a posição aleatória que deverá ser substituída) e a substituição FIFO (First in – First Out, que verifica qual foi o primeiro dado utilizado e o substitui pelo dado requisitado mais recentemente). A implementação da FIFO não foi feita em um header separado por motivos de choque de função. Pode-se verificar que sua implementação é feita no header “Cache.c” na parte de substituição no bloco cheio.

Funcionamento da cache:

É instanciado um vetor de x posições (nconjuntos) onde cada posição contém tamanho y (tbloco). O tamanho da posição interfere diretamente na quantidade de dados que poderão ser inseridos no endereço mapeado. Sendo assim, quando houver uma inserção deu um dado em uma posição x, a posição será carregada com um bloco de y dados pré-definidos. Ou seja, se a posição tem tamanho para 4 dados, estando ela vazia, quando houver a inserção de um dado, será carregado para a posição mais 3 dados.

- **Associatividade:**
  - **Direto:**

Após um dado ser solicitado pelo processador, ele é procurado no cache, se não houver esse dado, o mesmo é inserido na posição que foi mapeado. Se a posição mapeada estiver vazia o dado é simplesmente inserido, caso contrário, substitui-se o bloco.

  - **N-vias e Completamente Associativo:**

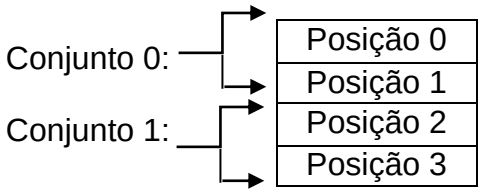
O vetor inicialmente contem n (nconjuntos) posições. Assim, se houver um aumento na associatividade as posições serão dividas em N conjuntos. Ou seja, em uma cache com 4 conjuntos e associatividade 2, o vetor será dividido em 8 conjuntos com duas posições cada conjunto.

Exemplo:

Direto:

Posição 0
Posição 1
Posição 2
Posição 3

2-vias:



Agora o endereço não será mais mapeado por posição e sim por conjunto. Como pode ser verificado no header “Calculo.c”.

Endereco = (indice % (NConjuntos/Associatividade)) * Associatividade;

Após isso, é verificado em qual posição do conjunto está o dado solicitado pelo processador. Caso não houver, será necessário inseri-lo, assim, verifica-se qual das posições está vazia. Se não houver posição vazia, utiliza-se de uma das politicas de substituição para saber em qual posição deve-se substituir o dado pelo mais recente solicitado.

Informaçãoes adicionais:

- Metodologia para execução do código:
    1. Descompactação do arquivo “Simulador_Cache”;
    2. Apertura do terminal na pasta que estão os arquivos descompactados;
    3. Para executar o código, digita-se “make” no terminal;
    4. Após a execução, insere-se a configuração do cache da seguinte maneira:

        <nconjuntos><tbloco><associatividade><logica_substituição>

        - 1 – nconjuntos: (2, 4, 8…).
        - 2 – tbloco: (2, 4, 8…).
        - 3 – associatividade: (2, 4, 8…).
        - 4 – logica_substituição: (R (random) ou F (FIFO)).

OBS: os dados de entrada estão no arquivo Teste.txt, que pode ser modificado. Os dados de entrada contidos nele devem ser em binário e conter 32 caracteres.