Universidade Federal de Ouro Preto - UFOP
Instituto de Ciências Exatas e Biológicas - ICEB
Departamento de Computação - DECOM
Ciência da Computação

Relatório TP1

BCC266 - Organização de Computadores

Mariana Macedo Santos, Gabriel Araújo Saldanha e Gustavo Zacarias Souza
Professor: Pedro Silva

Ouro Preto
16 de fevereiro de 2023

Sumário

Lista de Figuras

Lista de Códigos Fonte

1 Introdução

1.1 Especificações do problema

Precisamos construir uma máquina que será controlada por meio de instruções. A máquina possui memória e um programa que será interpretado. Além da memória principal (RAM), a máquina possui as memórias cache de níveis 1, 2 e 3. A memória conterá palavras do tipo inteiro, a memória principal é dividida em blocos e a memória cache é dividida em linhas. Além disso, simulamos o mapeamento associativo para troca de linhas entre as caches e a memória principal. Implementamos a política do FIFO(First In Firt Out), LFU(Least Frequent Used), LRU(Least Recently Used), mapeamento direto e mapeamento aleatório.

1.2 Considerações iniciais

Algumas ferramentas foram utilizadas durante a criação deste projeto:

- Ambiente de desenvolvimento do código fonte: vscode.
- Linguagem utilizada: C.
- Ambiente de desenvolvimento da documentação: Overleaf LaTeX. [1]

1.3 Implementação

Além da implementação das memórias cache de níveis 1 e 2, também implementamos a cache de nível 3 e alguns tipos de mapeamentos de memórias. Utilizamos os algoritmos de substituição: FIFO (first in first out), LFU (Least Frequently Used), LRU(Least Recently Used), mapeamento direto e o mapeamento aleatório.
No momento em que o processador precisar de um dado e ele estivar na memória cache, terá um cache hit, se não estiver, terá um cache miss. Utilizamos também contadores que facilitaram a informação de quantos hits e miss cada cache possuiu durante o processamento.
Na implementação do LFU, utilizamos um contador de uso para cada linha, e a partir deste método, temos a menos frequentemente usada.
Para o método FIFO, a nova informação da cache ficará na posição 0, todos os índices serão somados +1 e o ultimo índice (última linha da cache) passará para memória sucessora.
No LRU, utilizamos a variável de tempo, a que tiver maior tempo, será substituida.
O mapeamento direto foi implementado, utilizando o resultado do módulo do endereço e o tamanho da cache.
Já o mapeamento aleatório, o espaço da memória em que adicionaremos um novo dado será em uma posição aleatória.

```
1  int mapping(int address, Cache* cache, int Map_Type) {
2
3      int posicao = 0;
4      int size = cache->size;
5
6      switch (Map_Type) {
7          // Random
8          case 0:
9              posicao = rand() % cache->size;
10             break;
11         // FIFO
12         case 1:
13             posicao =  cache->size - 1;
14             break;
15         // LFU
16         case 2:
17             for(int i=0; i<size; i++)
18                 cache->lines[i].uso += 1;
```

[1] Disponível em https://www.overleaf.com/

```
19
20              int menor = cache->lines[0].uso;
21
22              for(int i=0; i<size; i++) {
23                  if(cache->lines[i].estaVazia == 1) {
24                      cache->lines[i].estaVazia = 0;
25                      return i;
26                  }
27                  if(menor > cache->lines[i].uso) {
28                      menor = cache->lines[i].uso;
29                      posicao = i;
30                  }
31              }
32              break;
33          // LRU
34          case 3:
35              for(int i=0; i<size; i++)
36                  cache->lines[i].tempo += 1;
37
38              int maior = cache->lines[0].tempo;
39
40              for(int i=0; i<size; i++) {
41                  if(cache->lines[i].estaVazia == 1) {
42                      cache->lines[i].estaVazia = 0;
43                      return i;
44                  }
45                  if(maior < cache->lines[i].tempo) {
46                      maior = cache->lines[i].tempo;
47                      posicao = i;
48                  }
49              }
50              break;
51          // Associativo
52          case 4:
53              posicao = address % cache->size;
54              break;
55      }
56
57      return posicao;
58  }
```

Código 1: Função do mapeamento.

A função mapping realiza o mapeamento de acordo com a opção inserida pelo usuário.
Ao final da execução, é impressa a tabela com a quantidade de hits e miss de cada cache.

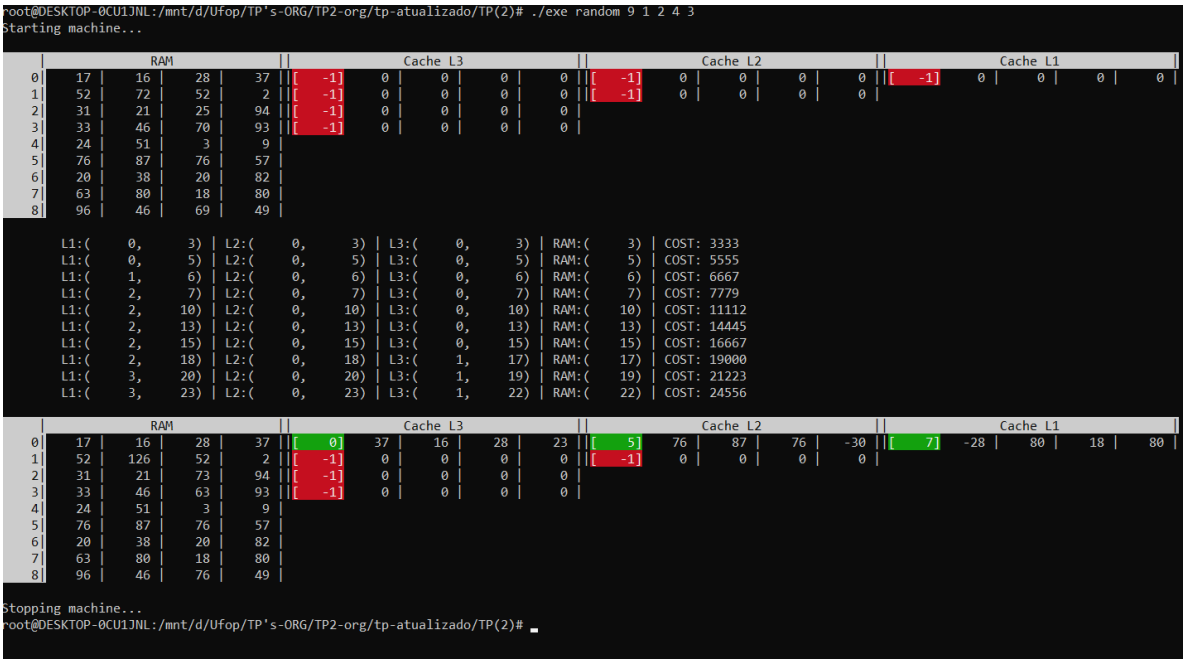

Figura 1: Um exemplo de saída.

1.4 Ferramentas utilizadas

Algumas ferramentas foram utilizadas para testar a implementação, como:

- *Valgrind*: ferramentas de análise dinâmica do código.

1.5 Especificações da máquina

A máquina onde o desenvolvimento e os testes foram realizados possui a seguinte configuração:

- Processador: i3 - 6ª Ger.
- Memória RAM: 4Gb.
- Sistema Operacional: Linux.

1.6 Instruções de compilação e execução

Compilando o projeto

Como utilizamos um arquivo MakeFile, para compilação do projeto, basta digitar: make

Para a execução do programa basta digitar:

./exe TIPO_INSTRUCAO [TAMANHO_RAM ou ARQUIVO_DE_INSTRUCOES] TAMANHO_L1 TAMANHO_L2 TIPO_DE_MAPEAMENTO

TIPO_INSTRUCAO: random; file;
TIPO_DE_MAPEAMENTO: 0 = random; 1 = FIFO; 2 = LFU; 3 = LRU; 4 = direto;

Exemplo:
./exe random 10 2 4 8 1

2 Impressões Gerais

Primeiramente, implementamos a memória cache de nível 3 . Isso envolveu a incluir a memória l3 na máquina, bem como o desenvolvimento de algoritmos para gerenciar o acesso aos dados armazenados na cache. Em seguida, implementamos os métodos de mapeamento LFU (Least Frequently Used), LRU (Least Recently Used) e FIFO (First In, First Out) para gerenciar a alocação de espaço na cache, o método será previamente escolhido pelo usuário. Esses métodos de mapeamento são comumente usados em sistemas de cache para decidir quais dados devem ser mantidos na cache e quais devem ser descartados para liberar espaço. A implementação desses métodos nos permitiu avaliar o desempenho da cache em diferentes situações.

3 Análise

Aprendemos que a hierarquia de memória se refere à organização de diferentes níveis de memória em um computador, desde a memória cache mais rápida até a memória principal mais lenta. O mapeamento de memória, por sua vez, se refere às técnicas usadas pelos computadores para acessar dados armazenados na memória de forma rápida e eficiente. Aprender sobre esses conceitos nos deu uma compreensão mais profunda de como os computadores funcionam e de como a organização da memória é importante para o desempenho geral do sistema.

4 Conclusão/Considerações Finais

Para a implementação deste trabalho era essencial entender o funcionamento de máquinas universais e sistemas de memória, particularmente com o sistema cache. Acrescentamos a cache de nível três, que está entre a principal e a de nível 2. Para o mapeamento associativo, implementamos funções de cada um dos métodos (LFU, FIFO, LRU, mapeamento associativo e aleatrio), que será chamado de acordo com a escolha do usuário ao iniciar o programa.
Encontramos nossa maior dificuldade quando estávamos escolhendo o método de substituição das linhas da cache. E desenvolvendo a lógica de cada um. Contudo, tudo foi ficando mais claro após vencermos este desafio.
Neste trabalho, adquirimos mais conhecimento de como manusear um programa em C, mas o mais importante, foi o conhecimento obtido de como ocorre o processamento de uma máquina e seu sistema de memória.