Trabalho de Arquitetura de Computadores

Leitura da Cache

|  |
| --- |
| Setor IV – Ciência e Tecnologia/Engenharia da Computação – UFRN 2016.1 |
| Aluno: Lukas Maximo Grilo Abreu Jardim |
| Professor: DIOGO PINHEIRO FERNANDES PEDROSA |

# Sumário

|  |  |
| --- | --- |
| Introdução | Página 3 |
| Análise do Experimento Parte 1 | Página 4 |
| Algorítmo do Exemplo | Página 6 |
| Análise do Experimento parte 2 | Página 9 |
| Algoritmo da Cache | Página 9 |
| Resultados e Conclusão | Página 13 |

# Introdução

A memória CASH, é um slot de memória que foi criada para ajudar a memória principal, servindo para armazenar programas em execução e melhorar o rendimento do computador.

|  |  |  |
| --- | --- | --- |
| Rótulo (TAG) | Linha (64 bytes) | Palavra |

Existem três métodos de acesso a uma cache: por mapeamento direto onde um determinado bloco sempre estará associado a uma mesma célula da cache; o mapeamento associativo – uma versão melhorada do mapeamento direto, onde ao invés de estabelecer células fixas na cache, visto no mapeamento direto, não determina um local fixo para interpretar os endereços da memória principal; e o mapeamento associativo por conjuntos que veio com o intuito de eliminar os problemas dos mapeamentos anteriores. O tipo de mapeamento utilizado foi o Mapeamento Associativo por Conjunto, onde a cache foi dividida em 64 conjuntos cada um com quatro linhas de cache mapeada por mapeamento associativo.

# Análise do experimento parte 1

Exemplo de processamento de uma linha de cache por mapeamento associativo por conjunto

0 | 0 | 4

1 | 0 | 4

2 | 0 | 4

3 | 0 | 4

4 | 0 | 4

5 | 0 | 4

6 | 0 | 4

7 | 0 | 4

8 | 0 | 4

9 | 0 | 4

10 | 0 | 4

11 | 0 | 4

12 | 0 | 4

13 | 0 | 4

14 | 0 | 4

15 | 0 | 4

0 | 4 | 4

F

0 | 4 | 5

1 | 0 | 5

2 | 0 | 5

3 | 0 | 5

4 | 0 | 5

5 | 0 | 5

6 | 0 | 5

7 | 0 | 5

8 | 0 | 5

9 | 0 | 5

10 | 0 | 5

11 | 0 | 5

12 | 0 | 5

13 | 0 | 5

14 | 0 | 5

15 | 0 | 5

1 | 5 | 5

F

0 | 4 | 9

1 | 5 | 9

2 | 0 | 9

3 | 0 | 9

4 | 0 | 9

5 | 0 | 9

6 | 0 | 9

7 | 0 | 9

8 | 0 | 9

9 | 0 | 9

10 | 0 | 9

11 | 0 | 9

12 | 0 | 9

13 | 0 | 9

14 | 0 | 9

15 | 0 | 9

2 | 9 | 9

F

0 | 4 | 5

1 | 5 | 5

5

Posição: 1

3 | 0 | 5

T

0 | 4 | 6

1 | 5 | 6

2 | 9 | 6

3 | 0 | 6

4 | 0 | 6

5 | 0 | 6

6 | 0 | 6

7 | 0 | 6

8 | 0 | 6

9 | 0 | 6

10 | 0 | 6

11 | 0 | 6

12 | 0 | 6

13 | 0 | 6

14 | 0 | 6

15 | 0 | 6

3 | 6 | 6

F

0 | 4 | 8

1 | 5 | 8

2 | 9 | 8

3 | 6 | 8

4 | 0 | 8

5 | 0 | 8

6 | 0 | 8

7 | 0 | 8

8 | 0 | 8

9 | 0 | 8

10 | 0 | 8

11 | 0 | 8

12 | 0 | 8

13 | 0 | 8

14 | 0 | 8

15 | 0 | 8

4 | 8 | 8

F

0 | 4 | 4

4

Posição: 0

5 | 0 | 4

T

0 | 4 | 2

1 | 5 | 2

2 | 9 | 2

3 | 6 | 2

4 | 8 | 2

5 | 0 | 2

6 | 0 | 2

7 | 0 | 2

8 | 0 | 2

9 | 0 | 2

10 | 0 | 2

11 | 0 | 2

12 | 0 | 2

13 | 0 | 2

14 | 0 | 2

15 | 0 | 2

5 | 2 | 2

F

No caso foi utilizado um vetor de 8 inteiros, e obtivemos:

Número de acertos = 2

Número de erros = 6

porcentagem de acertos = 0.25

porcentagem de erros = 0.75

A primeira coluna de cada linha representava a um bloco da linha da cache, enquanto a segunda linha mostrava o conteúdo do bloco. A terceira linha era a palavra a ser comparada com cada palavra armazenada no bloco da cache, portanto: se a palavra correspondesse a alguma palavra armazenada em algum bloco na cache, uma mensagem de acerto é contabilizada e não é necessário um novo bloco, ou então uma mensagem de falha é contabilizada.

## Algorítmo do Exemplo:

### Main

A classe usada no exemplo foi a cacheTeste, onde eu mandava um certo vetor de inteiros para a linha da cache representada por dois laços while e um laço if.

CacheDefin16KB a = **new** CacheDefin16KB();

Openfiles ab = **new** Openfiles();

String s = "A"

String yur[] = {"aqui"};

//int lines[] = {0};

System.***out***.println("Digite um limíte: ");

Scanner num = **new** Scanner(System.***in***);

**int** lim = num.nextInt();

**int** lines[] = **new** **int**[lim];

**int** c = 0;

System.***out***.println("Digite os endereços (números): ");

**while**(c < lim){

Scanner z = **new** Scanner(System.***in***);

lines[c] = z.nextInt();

c++;

}

c = 0;

**while**(c < lim){

System.***out***.print(lines[c] + " ");

c++;

}

System.***out***.print(lines.toString());

//System.out.println("digite um número menor que 16: ");

//Scanner lyy = new Scanner(System.in);

//int it = lyy.nextInt();

**int** i;

cacheTeste an = **new** cacheTeste();

//bool = an.boolConverter(it);

yur = an.leituraCache(lines);

### Função leituraCache

Alinha de cache é representada por 2 laços de repetição e um laço de condicional.

O primeiro laço de repetição era para controlar os outros dois, prosseguindo os laços até o o vetor terminar de ser mapeado. Os outros dois formam o restante da linha da cache: o if representa o bloco á ser acessado e o while que esta dentro do if é o algoritmo que realiza a parte principal do mapeamento associativo.

**int** c = 0, i = 0, lcount = 0, ercount = 0, trans = 0;

c = 0;

**while**(c < nline.length){

System.***out***.print(nline[c] + " ");

c++;

}

System.***out***.println(c);

c = 0;

**int** cach1[] = **new** **int**[16];

**int** res1[] = **new** **int**[16];

String resultCache[] = **new** String[nline.length];

**while**(i < nline.length){

resultCache[i] = "resultado a definir";

System.***out***.print("-");

i++;

}

i = 0;

**while**(c < 16){

cach1[c] = 0;

res1[c] = 0;

System.***out***.println(cach1[c] + " " + c + " " + res1[c]);

c++;

}

c = 0;

**int** j = 0;

//Mapeamento Associativo

**while**(i < nline.length){

**if**(c < 16){

**while**(j < 16){

System.***out***.println(j + " | " + cach1[j] + " | " + nline[i]);

**if**(nline[i] == cach1[j]){

resultCache[i] = "T";

trans = j;

System.***out***.println(cach1[j]);

res1[c] = 2;

lcount++;

}

**if**(res1[c] == 2)

**break**;

j++;

}

j = 0;

**if**(res1[c] == 0){

cach1[c] = nline[i];

resultCache[i] = "F";

ercount++;

System.***out***.println(c + " | " + cach1[c] + " | " + nline[i]);

}**else**{

System.***out***.println("Posição: " + trans);

System.***out***.println(c + " | " + cach1[c] + " | " + nline[i]);

//c--;

}

System.***out***.println(resultCache[i]);

c++;

**if**(res1[c - 1] != 0){

c--;

res1[c] = 0;

}

}

//miltonlqueiroz@hotmail.com

**if**(res1[c - 1] == 2)

//c--;

**if**(c == 16)

c = 0;

trans = 0;

i++;

}

System.***out***.println("Número de acertos = " + lcount);

System.***out***.println("Número de erros = " + ercount);

**float** total = lcount + ercount;

**float** er = ercount;

**float** ar = lcount;

**float** erp = er/total;

**float** arp = ar/total;

System.***out***.println("porcentagem de acertos = " + arp);

System.***out***.println("porcentagem de erros = " + erp);

# Análise do experimento parte 2

A Cache Principal

Na cache principal, são 64 conjuntos de 4 linhas de cache cada organizadas por mapeamento associativo. Nesse caso foram precisas mais duas classes a primeira com a função definitiva da cache e duas funções para ler os arquivos .dat e .asc respectivamente:

No caso do arquivo .dat, foi retirado o numero de linhas para então elaborar o vetor de inteiros que irá ser utilizado pelo sistema da cache agora com mais três laços while antes do primeiro if (dentro do while do vetor fornecido): o primeiro para representar os conjuntos, o segundo para repreentar as Tags e o terceiro para representar as linhas da cache.

## Algoritmo da Cache

### Main (continuação)

int i

i = a.OpenAsc();

int[] h = ab.Storeadress(i);

a.CachSim(h);

### OpenAsc (A função vai abrir a .dat e pegar o número de linhas do arquivo)

**public** **int** OpenAsc(){

File dir = **new** File("C:/Users/cliente/Documents/ECT UFRN 2016.1/Eclipse/tarefa mapeamento");

**boolean** statusDir = dir.mkdir();

System.***out***.print(statusDir);

File file = **new** File(dir,"acessos.dat");

**try**{

FileReader read = **new** FileReader(file);

BufferedReader buff = **new** BufferedReader(read);

String line = "";

//String tab[];

**int** i = 0;

**while**((line = buff.readLine())!= **null**){

System.***out***.println(line);

//tab[i] = line;

i++;

}

//String tab1[] = new String[i];

String linerd[] = **new** String[i];

**return** i;

}**catch**(IOException e){

e.printStackTrace();

**return** 0;

}

}

### Storeadress (A função vai obter o vetor de endereços a partir do inteiro gerado pelo OpenAsc e pelo arquivo.dat)

**public** **int**[] Storeadress(**int** i){

**int** d[] = **new** **int**[i];

**int**[] nule = **new** **int**[i];

File dir = **new** File("C:/Users/cliente/Documents/ECT UFRN 2016.1/Eclipse/tarefa mapeamento");

**boolean** statusDir = dir.mkdir();

System.***out***.print(statusDir);

File file = **new** File(dir,"acessos.dat");

**try**{

FileReader read = **new** FileReader(file);

BufferedReader buff = **new** BufferedReader(read);

String line = "";

//String tab[];

**int** c = 0;

**while**(((line = buff.readLine())!= **null**)&&(c < d.length)){

System.***out***.println(line);

d[c] = line.hashCode();

System.***out***.println(d[c]);

nule[c] = 0;

c++;

}

//String tab1[] = new String[i];

//int linerd[] = new int[i];

**return** d;

}**catch**(IOException e){

e.printStackTrace();

**return** nule;

}

}

### CacheSim (função que realizará o mapeamento previsto)

**public** String[] CachSim(**int**[] nline){

//String result[] = {"A definir...."};

//int liml = nline.length;

//int limc = ncoln.length;

//int lim;

//int cach[] = {0};

//int cach[] = new int[liml];

/\*

\* 256 linhas;

\* cada linha ha 64 bytes;

\* cada linha armasena 16 int;

\* cache dividida em conjuno;

\* 64 conjuntos cada um com 4 linhas;

\*/

System.***out***.println(nline.length);

**int** c = 0, i = 0, lcount = 0, ercount = 0, trans = 0;

**int** conjunto = 0; **int** cachline = 0; **int** cachtag = 0;

/\*if(liml < limc)

lim = limc;

else

lim = liml;

\*/

c = 0;

**while**(c < nline.length){

System.***out***.print(nline[c] + " ");

c++;

}

System.***out***.println(c);

c = 0;

**int** cach1[][] = **new** **int**[4][16];

**int** res1[] = **new** **int**[16];

String resultCache[] = **new** String[nline.length];

**while**(i < nline.length){

resultCache[i] = "resultado a definir";

System.***out***.print("-");

i++;

}

i = 0;

**while**(i < 4){

**while**(c < 16){

cach1[i][c] = 0;

res1[c] = 0;

System.***out***.println(cach1[i][c] + " " + c + " " + res1[c]);

c++;

}

i++;

}

c = 0;

i = 0;

**int** j = 0, tag = 0;

//Mapeamento Associativo

**while**(i < nline.length){

**while** (conjunto < 64){

**while** (cachtag < 4){

**while** (cachline < 16){

**if**(c < 16){

**while**(j < 16){

**if**(i == nline.length)

**break**;

System.***out***.println(j + " | " + cach1[cachtag][j] + " | " + nline[i] + "|" + cachtag);

**if**(nline[i] == cach1[cachtag][j]){

resultCache[i] = "T";

trans = j;

System.***out***.println(cach1[cachtag][j]);

res1[c] = 2;

lcount++;

}

**if**(res1[c] == 2)

**break**;

j++;

}

j = 0;

**if**(i < nline.length){

**if**(res1[c] == 0){

cach1[cachtag][c] = nline[i];

resultCache[i] = "F";

ercount++;

System.***out***.println(c + " | " + cach1[cachtag][c] + " | " + nline[i] + "|" + cachtag);

}**else**{

System.***out***.println("Posição: " + trans);

System.***out***.println(c + " | " + cach1[cachtag][c] + " | " + nline[i] + "|" + cachtag);

//c--;

}

System.***out***.println(resultCache[i]);

}

c++;

**if**(res1[c - 1] != 0){

c--;

cachline--;

res1[c] = 0;

}

}

**if**(i == nline.length)

**break**;

cachline++;

**if**(c == 16)

c = 0;

trans = 0;

i++;

}

**if**(i == nline.length)

**break**;

**if**(cachline == 16)

cachline = 0;

cachtag++;

}

**if**(i == nline.length)

**break**;

**if**(cachtag == 4)

cachtag = 0;

conjunto++;

}

**if**(conjunto == 64)

conjunto = 0;

}

System.***out***.println("Número de acertos = " + lcount);

System.***out***.println("Número de erros = " + ercount);

**float** total = lcount + ercount;

**float** er = ercount;

**float** ar = lcount;

**float** erp = er/total;

**float** arp = ar/total;

System.***out***.println("porcentagem de acertos = " + arp);

System.***out***.println("porcentagem de erros = " + erp);

**return** resultCache;

}

Com tudo isso obteve-se:

Número de acertos = 0

Número de erros = 10000

porcentagem de acertos = 0.0

porcentagem de erros = 1.0

# Resultados e Conclusão

Esse resultado foi obtido com o fato de que o mapeamento por conjunto realizado não tem 100% de perfeição devido a um laço while sobressalente e mesmo que tivesse a taxa de acertos ainda seria pequena dependendo dos endereços que a função acessaria. Foi possível ler e exibir o arquivo .asc, mas a classe Java para transformar o código ASCII em uma imagem normal e exibi-la ainda é desconhecida.