Lab 5 – Arquiteutra de Computadores

Teresa Nogueira 100029 & Inês Paiva 99961

Exercício 1

4.

Elemento	Endereço (Hexadecimal)
A[32,24]	0x10002060
B[32,24]	0x10006060
C[32,24]	0x1000a060
D[32,24]	0x1000e060
T1[32,24]	0x10012060
T2[32,24]	0x10016060

6.

Tag	Index	Offset
22	6	4

7.

Elemento	Endereço (Hexadecimal)	Tag	Index	Offset
A[32,24]	0x10002060	10002h 00	00 0110	0000
B[32,24]	0x10006060	10006h 00	00 0110	0000
C[32,24]	0x1000a060	1000ah 00	00 0110	0000
D[32,24]	0x1000e060	1000eh 00	00 0110	0000
T1[32,24]	0x10012060	10012h 00	00 0110	0000
T2[32,24]	0x10016060	10016h 00	00 0110	0000

Q1.8.

O número de vias de associatividade da cache na sua generalidade diminui o número de conflitos na cache. Tal é verificado quando entramos na fase de loop do programa, não existindo um número de vias suficiente levaria aos dados serem substituidos na cache (por exemplo, quando os indexs são os mesmos mas as tags diferentes, ou seja, são dois vetores diferentes, com um menor número de vias, seria necessária a troca quase constante dos valores na cache algo que não acontece aumentando o número de vias, pois para o mesmo index existem n vias disponíveis), o que por relação, levaria a uma maior taxa de Miss rate.

Exercício 2

5.

Nº de Hits:	256434	Nº total de acessos à memória:	540675
Nº de Misses:	284241	Miss-Rate (total) [%]:	52,57

6.

$$TAcesso = TL1 + MRL1 \times TRAM$$

Tempo médio de cada acesso [ns]:	53,57 ns
Tempo total de acesso à memória [ms]:	28,963 ms

7.

Tempo médio de cada acesso [ns]:	100 ns
Tempo total de acesso à memória [ms]:	54,0675ms

Q2.8.

Os tempos de acesso no caso de o sistema não incorporar a cache são quase 50% mais lentos do que quando esta está implementada. Esta discrepância de valores deve-se ao facto de o acesso á memória ser um processo muito lento e por isso, quando é implementada uma cache os acessos á memória diminuem, uma vez que a cache guarda os valores por linhas, logo para cada Miss, ou seja acesso á memória, vai também buscar os valores seguintes de dados diminuindo o número de acessos á memória principal que tem um tempo de acesso muito superior ao acesso da cache.

Exercício 3

Q3.2.

É esperado que o resultado seja idêntico ao obtido pelo método anterior, já que apenas se aterar a forma de resolver o mesmo problema.

Q3.3.

A eventual troca de ordem dos dois ciclos for garante uma maior taxa de sucesso, uma vez que a cache quando vai à memória “carrega” a sua linha por inteiro logo se o loop verificar primeiro as linhas das matrizes o número de misses diminuiu uma vez que esses valores são logo carregados da memória quando se dá o miss por passagem para a linha seguinte. Assim o miss-rate é inferior nesse caso.

Q3.4.

Tal como indicado anteriormente, ao aceder à memória é carregado para a cache os valores para preencher uma linha da cache, face a isto, ao fazer o loop tal que matrizes sejam lidas linha a linha vai diminuir o miss-rate sendo por isso mais favorável.

Q3.5.

	Ordem i,j	Ordem j,i
Nº de acessos à memória (total):	540675	540675
Miss-Rate (total) [%]:	50,86	52,57
Tempo médio de cada acesso [ns]:	51,86	53,57
Tempo total de acesso à memória [ms]:	28,039	28,963

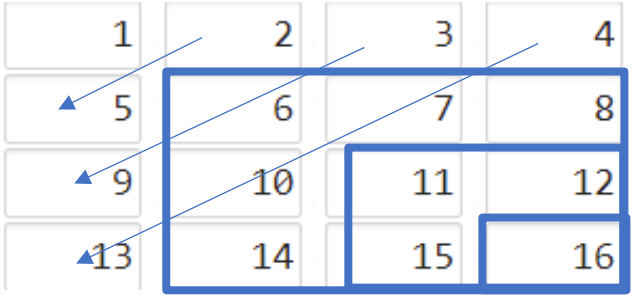
Q3.6. Podemos inferir que, pelo explicitado na pergunta 3.3 que era esperado obter mais ganhos face à versão original, por outro lado, tendo em conta a diminuição do miss rate após a alteração do código na segunda parte, podemos inferir que a maior quantidade de misses se encontra na primeira parte do programa.

Q4.1.

```
# inicializa?ao de auxiliares:
          addi t6, zero, 4 #
        mul a3, s9, t6   # a3 -> 4*N
        add a1, zero, s2 # a1 -> *B

        # for(i=0){
        add t1, zero, zero
        # for(j=0,...,...){
        add t2, zero, zero
transposta2:
          add t0, zero, a1 # t0 -> *auxB = B

          beq t1, t2, else
          lw t3, 0(a1)
          mul t4,t2,t6
          mul t5,a3,t2
          sub t4,t5,t4 #ajuste do endere♦o 64*j*4-j*4
          add a1,a1,t4
          lw t4, 0(a1)
          sw t3, 0(a1)
          mv a1,t0
          sw t4,0(a1)
else:
          addi t2,t2,1
          addi a1,a1,4
          blt t2,s9, transposta2

          addi s9,s9,-1
          mv t2,x0
          addi a6,a6,4
          add a1,a1,a6
          bne s7,s9, transposta2
```

```
1  for(i = 0; i<=N; i++)
2  {
3      for(j = 0; j<=N; j++)
4      {
5          if(i != j)
6          {
7              B[i][j] =  B[j][i];
8          }
9          Else
10         {
11             continue;
12         }
13     }
14 }
```

Figura 1 – ideia inicial



Q4.3.

Nº de Hits:	466446	Nº total de acessos à memória:	548742
Nº de Misses:	82297	Miss-Rate (total) [%]:	15

Q4.4. Verificamos um decréscimo do miss-rate embora um aumento do número de acessos à memória, uma vez que ao fazer a transposição da matriz e multiplicando as matrizes através da

transposta, quando vamos à memória principal e esta carrega valores para a cache, vai preencher toda uma coluna da matriz, logo, uma vez que a multiplicação é feita por linha*coluna, o número de misses irá diminuir, é aproveitada a localidade espacial da cache.

Q4.5.

Tempo médio de cada acesso [ns]:	16 ns
Tempo total de acesso à memória [ms]:	8,7799 ms
Speedup:	3,194

Q5.2

Nº de Hits:	431057	Nº total de acessos à memória:	548742
Nº de Misses:	117685	Miss-Rate (total) [%]:	21,45

Tempo médio de cada acesso [ns]:	22,45 ns
Tempo total de acesso à memória [ms]:	11,7705 ms
Speedup:	0,7459

Q5.3

Apesar de se usar a matriz trasposta para reduzir o miss-rate, ao reduzir o número de vias os vários indexs vão-se sobrepor mas com tags diferentes e uma vez que já só existe uma via o número de misses vai aumentar, pois é necessário para cada index ir trocando os dados na cache dependendo qual é a matriz que se está a utilizar, tal se verifica com mais proeminência neste excerto de código:

```
f1loop3:
        # ... += memory(A,R,i,k,0)*...
        lw t5, 0(a0)
        # ... += ...*memory(B,R,k,j,0)
        lw t6, 0(a1)
        # ... += memory(A,R,i,k,0)*memory(B,R,k,j,0)
        mul t5, t5, t6
        # aux1 += memory(A,R,i,k,0)*memory(B,R,k,j,0)
        add t4, t4, t5
        # A++
        addi a0, a0, 4
        # auxB += 4 (salta uma coluna)
        addi a1, a1, 4
        # for(...,...,k++)
        addi t3, t3, 1
        # for(...,k<size,...)
        blt t3, s9, f1loop3
        # }
 # }
```

Exercício 6

Q6.3

Nº de Hits:	465517	Nº total de acessos à memória:	536550
Nº de Misses:	71033	Miss-Rate (total) [%]:	13,14

Q6.4. Verifica-se um ligeiro decréscimo no miss-rate o que leva a um melhor desempenho e da utilização das capacidades da cache. Para tal foi realizada uma junção dos dois códigos denominados por first part e second part, descrito pelo código c abaixo:

```
for (i=0;i<N;i++)
{
   for (j=0;j<N;j++){
       aux1=0;
           for (k=0;k<N;k++){

               T1[i][j] += A[i][k] * B[k][j];
           }
           D[i][j] = alpha*T1[i][j] + beta*C[i][j];
   }
}
```

Ainda foi alterado o endereço da matriz T1 de forma a que tivesse o mesmo index que a segunda linha de A através de uma matriz auxiliar, por sugestão do professor Paulo Lopes:

```
NULL:    .zero 256
```

Q6.5

Tempo médio de cada acesso [ns]:	14,14
Tempo total de acesso à memória [ms]:	7,587
Speedup:	3,6957

Avaliamos o ganho através do speedup.

Q7.

$TAcesso = TL1 + MRL1 \times (TL2 + MRL2 \times TRAM)$

Tempo médio de cada acesso [ns]:	16,267
Tempo total de acesso à memória [ms]:	8,795
Speedup:	3,188