Parte 1

1. O computador que usámos foi o lab6p7.
   Com um stride de 2048, podemos observar que até 32KB temos um tempo de acesso praticamente constante, sendo que de 32KB para 64KB existe um salto relativamente mais íngreme que entre 4, 8, e 32KB e entre 64KB e 128KB.
   Assim, podemos concluir que o tamanho da cache é 32KB.

   Tem 32KB (com stride de 2048, vemos claramente que entre
   32KB e 64KB há um salto relativamente mais ingreme que entre
   4, 8, e 32KB e entre 64KB e 128KB)

2. Através da figura 1, podemos ver que, com um stride de 4, entre 4KB e 64KB está tudo a dar *overlap*, mas a partir de 128KB o tempo de execução é substancialmente maior, que corresponde às *miss penalties*. Podemos então assumir que até 64KB todos os arrays podem ser guardados na cache, na sua totalidade.
   Portanto, a capacidade da cache do computador é de 64KB.

   64KB -> entre 4KB a 64KB quando o stride é 4 está tudo a dar overlap,
   mas 128KB já demora quase o dobro do que 64KB com um stride de 4.
   Isto indica que em princípio a cache tem uma capacidade de 64KB.

3. Ao observar, por exemplo, a curva de 4MB, o tempo de acesso estabiliza entre um stride de 16 e 32, o que apenas acontece quando o stride é igual ou maior que o tamanho do bloco da cache, isto é, a partir de 16 o stride já é suficiente para saltar blocos inteiros. Logo, cada bloco da cache tem um tamanho de 16 Bytes.

   16 Bytes por bloco. Ao observar, por exemplo, a curva de 4MB, o tempo
   claramente estabiliza entre um stride de 16 e 32, indicando que a partir
   de 16 o stride já é suficiente para saltar blocos inteiro, ou seja,
   é o tamanho exato do bloco.

4. Para estimar o tempo de 2 hits (1 read e 1 write), basta considerar o tempo mais rápido registrado no gráfico. Este foi cerca de 330 ns ao percorrer o array de 64k com um stride de 256 bytes. Já para a estimação do tempo de 1 miss e 1 hit (ler do array causa um miss, mas o write a seguir já causará um hit), basta observar o tempo que demora a fazer read e write ao iterar um array maior que a cache com um stride maior que um bloco, visto que aí o miss no read é garantido. Sendo assim, considerando o array de 4M e de um stride de 32 Bytes, $t_{miss}+t_{hit}=1000ns$. Juntando estes dois factos, $2t_{hit}=330ns$ e $t_{miss}+t_{hit}=1000ns$, temos que $t_{miss}+t_{hit}=t_{hit}+t_{penalty}+t_{hit}=t_{penalty}+2t_{hit}=1000ns \Rightarrow t_{penalty}=1000-2t_{hit}=1000-330=670ns$, ou seja, que a miss penalty é cerca de 670ns.

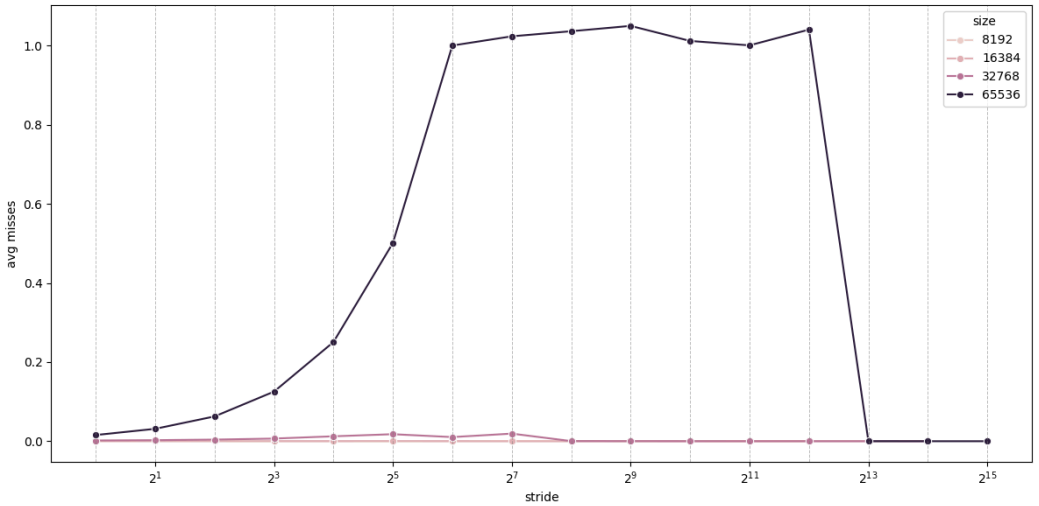
Parte 2

a) O único evento que será analisado durante a execução é o L1_DCM, ou seja, o L1 data cache miss. Como o seu nome indica, este evento representa um miss num acesso à cache de dados.

c) Considerando strides maiores, como 2048 Bytes por exemplo, caso uma dada porção caiba toda em memória, apesar de causar muitos misses na primeira iteração, as iterações restantes vão causar todas hit, pelo que os average misses devem de ser relativamente baixos. Caso a porção não caiba em cache, um stride de 2048 Bytes, que em princípio será maior que o tamanho do bloco, deverá causar sempre misses, pelo que os average misses serão relativamente altos. Ao passar de 32KB para 64há claramente um salto nos average misses, ao iterar com um stride de 2048 Bytes, indicando assim que, muito provavelmente, a cache L1 terá 32KB.

Cada bloco da cache L1 tem 64 Bytes. Ao observar a curva da porção de 64KB, é possível observar que os average misses vão aumentando lentamente indicando que há cada vez menos hits dentro de cada bloco, devido ao stride ir aumentando, porém entre um stride de 64 Bytes e 128 Bytes já não há diferença, demonstrando assim que o stride já é maior que um bloco em ambos os casos, pelo que o tamanho do bloco tem de ser de 64 Bytes.

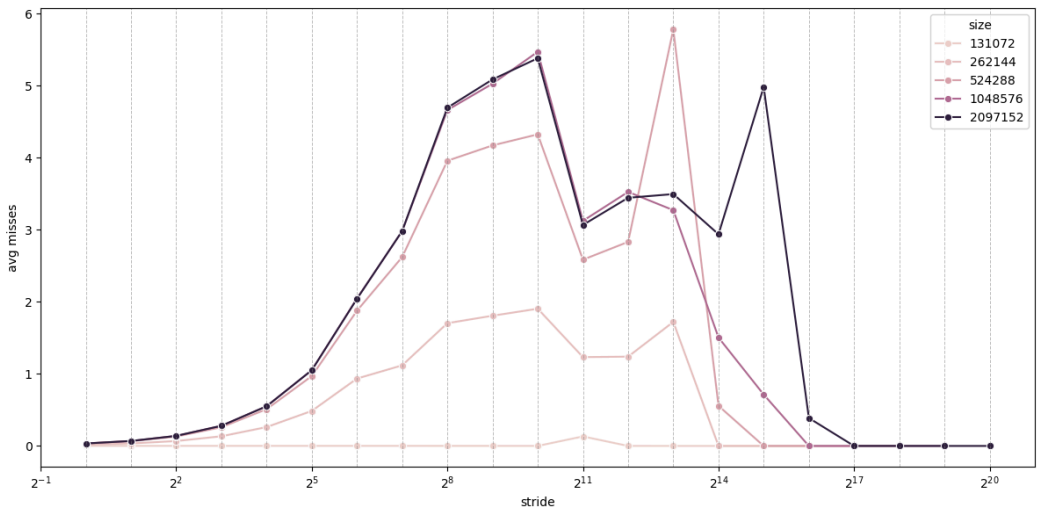
Ao considerar o array de 64KB, é possível observar que para strides grandes há sempre um miss rate de 100%, porém quando chega ao stride de 2^13 o miss rate cai diretamente para 0%. Isto acontece por causa da associatividade da cache. Ao analisar os acessos gerados, conclui-se que este fenómeno só é possível se a cache L1 tiver 8 vias. Se tivesse quatro vias, haveria 8 acessos no total, incidentes num só bloco, pelo que a miss rate seria de 100% à mesma. Se tivesse 16 vias, o stride de 2^12 devia de apresentar uma miss rate de 0%, pois haveria 16 acessos, também num só bloco. Assim, considerando que a cache tem 32KB e blocos de 64B, a cache tem de ter 8 vias de associatividade.



Cache L2:
Size: Considerando um stride de 1024 Bytes, por exemplo, é possível observar que há um grande salto em termos de miss rate entre o array de 128KB e 256KB e ainda outro grande salto entre o de 256KB e os arrays de 512KB, 1MB e 2MB. Sendo assim, podemos concluir que o cache L2 deve ter 256KB, visto que não apresenta uma miss rate tão alta como os três maiores arrays, indicando assim que o array de 256KB ainda cabia totalmente em cache . A miss rate que este tamanho de array apresenta deve-se, provavelmente, a erros nas medições devido, por exemplo, à intervenção do sistema operativo.

Block Size: Analisando apenas os tamanhos de array maiores que a cache (512KB, 1MB e 2MB), podemos ver que a miss rate é muito errática, pelo que é difícil determinar o tamanho do bloco com grande precisão. Ainda assim, tal como com a cache L1, a miss rate vai aumentando gradualmente, isto porque, enquanto o stride for menor que o tamanho do bloco, haverá sempre alguns hits na cache. Contudo, a partir de um stride de 256 Bytes, a miss rate estabiliza um bocado, indicando que já se está a saltar blocos completos. Sendo assim, é esperado que cada bloco seja de 256 Bytes.

Associatividade: Analisando apenas o array de 2MB, que não cabe em cache, podemos observar que a partir de um stride de 2^{17} Bytes, a miss rate vai para 0%. Isto deve-se à associatividade da cache. Tal como foi feito com a cache L1, considerando uma cache L2 com 256KB e blocos de 256 Bytes, é possível concluir que esta tem 16 vias de associatividade. Isto porque, ao analisar os $2^{21}/2^{17}=16$ acessos efetuados ao percorrer o array de 2MB com um stride de 2^{17} Bytes, se a cache L2 tivesse menos do que 16 vias, então haveria substituição de blocos dentro de conjuntos, pelo que a miss rate não seria nula. Porém, caso houvessem mais do que 16 vias, o stride de 2^{16} Bytes também deveria de apresentar uma miss rate de 0%. Como tal não é o caso, a única maneira de obter o gráfico obtido, assumindo que os outros dois parâmetros estão corretos, é se a cache L2 tiver 16 vias de associatividade.



Parte 3

3.2
3.2.1 MM1
a) 16bit integer, N = 512, m1: NxN, m2: NxN, m3:NxN
Como cada elemento da matriz é um inteiro de 16 bits, cada elemento 2 bytes
Logo cada matriz ocupa: 512*512*2 bytes = 524288 bytes (2^9*2^9*2 = 2^{19} bytes)

b)
Total number of L1 data cache misses: 135.350767 ×10^6
Total number of load/store instructions completed: (402.654039 + 134.218057) ×10^6 =
536.872095 ×10^6
Total number of clock cycles: 591.546446 ×10^6
Elapsed time: 0.197181 seconds

c)
L1_hit_rate= 1 - L1_miss_rate = 1 - misses/total_instrucs = 1 - 135.428416 ×10^6/536.872095 ×10^6 = 0.74774547371 -> 74.77 %

3.2.2 (MM2)
a)
Total number of L1 data cache misses: 4.216631 ×10^6
Total number of load/store instructions completed: (402.654021 + 134.218057)×10^6 =
536.872078×10^6
Total number of clock cycles: 519.699357 ×10^6
Elapsed time: 0.173234 seconds

b)
L1_hit_rate = 1 - misses/total_cache_access(instructions) = 1 - 4.218997 ×10^6/536.872078×10^6 = 0.99214152277
c)
Total number of L1 data cache misses: 4.484481×10^6
Total number of load/store instructions completed: (402.916903 + 135.004586)×10^6 =
537,921489 ×10^6
Total number of clock cycles: 536.876825 ×10^6
Elapsed time: 0.178960 seconds

Comment on the obtained results when including the matrix transposition in the execution time:
Ao incluir o tempo gasto na transposição da matriz no tempo de execução, podemos observar que este tempo é praticamente negligenciável. (...)

d)
$\Delta HitRate = hitRate_{mm2} - hitRate_{mm1} = 0.991663317 - 0.747890106 = 0,243773211$
$Speedup(\#Clocks) = \#Clocks_{mm1} / \#Clocks_{mm2} = 591.546446 \times 10\text{^}6/537,921489 \times 10\text{^}6 = 1.1$

$Speedup(Time) = Time_{mm1} / Time_{mm2} = 0.197181/0.178960 = 1.1$

A hit rate melhorou muito, o que era de esperar uma vez que passa a haver muitos mais acessos contíguos. Na primeira versão, havia sempre pelo menos um cache miss por cada iteração do loop interior, dado que uma das matrizes era percorrida linha a linha em vez de coluna a coluna. Já na segunda iteração, todas as matrizes são percorridas coluna a coluna, pelo que passa a haver misses apenas quando se chega ao fim de um bloco, em vez de isso acontecer todas as iterações.

Apesar disso, o speedup, tanto em termos de tempo como ciclos, não foi tão impressionante. Porém, tal também não é grande surpresa pois só se melhorou um dos vários aspetos do programa e, consequentemente, o efeito não é tão impactante.

No geral, caso não existam limitações de memória no sistema, a otimização parece ainda assim valer a pena, uma vez que, mesmo considerando o custo de transposição, o algoritmo passa a ser mais rápido.

3.2.3

a) Anteriormente, foi calculado que os blocos eram de 64 Bytes. Assim, como cada matriz é composta por inteiros de 16 bits (2 bytes), cada bloco acomoda 32 elementos.

b) MM3 p 64B

Total number of L1 data cache misses: 3.114485 ×10^6

Total number of load/store instructions completed: (403.317486 + 134.627678) ×10^6 = 537.945164×10^6

Total number of clock cycles: 217.563744 ×10^6

Elapsed time: 0.072522 seconds

c) L1_hit_rate = 1 - misses/total_ins = 1 - 3.114485 ×10^6/537.945164×10^6 = 0.99421040431

d)

$\Delta HitRate = hitRate_{mm3} - hirRate_{mm1} = 0.99421040431 - 0.747890106 = 0.246320298$

$Speedup(\#Clocks) = \#Clocks_{mm1} / \#Clocks_{mm3} = 591.546446\times10\text{^}6/217,563744 \times10\text{^}6 = 2.72$

A implementação com sub matrizes apresenta uma hit rate muito superior comparativamente à implementação básica. Isto era de esperar, visto que o padrão de acesso às matrizes na nova implementação é otimizado de maneira a aumentar a localidade espacial dos acessos.

O speedup também é muito maior, sendo a nova implementação aproximadamente 2.72 vezes mais rápida que a versão anterior.

Desta maneira, é possível concluir que esta otimização foi bem sucedida, visto que apresenta os melhores resultados de entre as três versões demonstradas e ainda tem o benefício de não necessitar memória extra como a segunda implementação apresentada.

e)

$$\Delta HitRate = hitRate_{mm3} - hitRate_{mm2} = 0.99421040431 - 0.991663317 = 0.002547087$$

$$Speedup(\#Clocks) = \#Clocks_{mm2} / \#Clocks_{mm3} = 537,921489 \times 10\text{^}6/217,563744 \times 10\text{^}6 = 2.47$$

A implementação com sub matrizes apresenta um speedup de cerca de 2.47 vezes relativamente à implementação que transpõe a matriz da direita, apesar das hit rates serem praticamente iguais.

Apesar da diferença entre hit rates não ser negativa, como descrito no enunciado, o speedup positivo pode ser explicado pelas hit rates nas caches de nível dois. A terceira implementação apresentou 1.315750 * 10^6 acessos à cache L2 e 2.291276 * 10^6 misses, enquanto que a segunda acedeu 2.570868 * 10^6 vezes à cache L2, causando 9.092027 * 10^6 misses. Como a segunda implementação acedeu só o dobro das vezes à cache L2, comparativamente à terceira implementação, mas causou por volta de quatro vezes mais misses, explicando assim o porquê da nova implementação ser muito mais rápida.

Considerando que um acesso à cache L2 implica uma penalização muito grande em termos de ciclos, não é surpreendente que o speedup seja positivo apesar das hit rates na cache L1 serem quase iguais.

3.2.3

O comando lscpu indica que a cache de dados L1 tem 32KB, tal como previsto, e que a cache L2, que é partilhada, tem 256KB, assim como tinha sido calculado anteriormente. Para além disso, o comando papi_mem_info também indica que o tamanho do bloco e a associatividade da cache L1 estavam corretos, sendo estes 64 Bytes e 8 vias, respectivamente. Porém, a cache L2 tem blocos de 64 Bytes e 4 vias de associatividade, ao contrário dos 256 Bytes e 16 vias medidos. Este erro aconteceu, muito provavelmente, devido a medições incorretas da cache L2, visto que o gráfico da mesma apresenta miss rates para além dos 100%. Suspeitamos que algum destes erros surjam diretamente do PAPI, uma vez que é incapaz de medir os misses da cache L2 diretamente em hardware. Outra possível fonte de erros é o sistema operativo que pode trocar processos muito intensivos por outros, invalidando assim a cache.