First Lab Assignment: System Modeling and Profiling

STUDENTS' IDENTIFICATION:

Number:	Name:
ist1110633	Filipe Oliveira
ist1110720	Francisco Andrade
ist1116206	Luiz Silva

2 First steps

Please justify all your answers with values from the experiments.

1. What is the cache capacity of the computer you used (please write the workstation name)?

Array Size (KiB)	8	16	32	64	128	256
t2-t1 (s)	0.001777	0.003610	0.007116	0.016130	0.037211	0.086599
# accesses a[i]	819200	1638400	3276800	6553600	13107200	26214400
# mean access time (ns)	2.169217	2.202517	2.182533	2.466532	2.832689	3.301921

Ao usar o lab1p6, foi possível determinar que a capacidade da cache é de **32 kB**. A análise da tabela de resultados mostra que o tempo médio de acesso se mantém relativamente baixo e constante para tamanhos de array até 32 kB, variando entre 2.17 ns e 2.20 ns. Este comportamento indica que, para estes tamanhos, o array cabe confortavelmente na cache L1, resultando numa elevada taxa de acertos. No entanto, quando o tamanho do array aumenta de 32 kB para 64 kB, observa-se um aumento notório no tempo médio de acesso (de 2.18 ns para 2.46 ns), e este aumento continua a acentuar-se para tamanhos maiores. Este salto no tempo de acesso sugere um aumento significativo na taxa de falhas, pois o array já não cabe na cache L1. Estas falhas forçam o processador a aceder a níveis de memória mais lentos, como a cache L2, o que justifica o aumento do tempo de acesso. Logo, o ponto de inflexão no desempenho entre 32 kB e 64 kB permite concluir que a capacidade da cache L1 é de **32 kB**.

Consider the data presented in Figure 1. Answer the following questions (2, 3, 4) about the machine used to generate that data.

2. What is the cache capacity?

A análise da Figura 1 revela dois grupos distintos de tempos de acesso, um em torno de 375 ns e outro em torno de 900 ns. O primeiro grupo, com tempos de acesso mais baixos, corresponde a tamanhos de array que cabem inteiramente na cache. O segundo grupo, com tempos de acesso mais elevados, representa arrays que excedem a capacidade da cache, resultando num aumento da penalidade por *misses*. O ponto de inflexão, onde o tempo de acesso aumenta abruptamente, ocorre após o tamanho de array de 64 kB. Isto indica que arrays até 64 kB beneficiam de uma alta taxa de acertos, enquanto arrays maiores (como 128 kB) sofrem de *capacity misses*, forçando acessos a níveis de memória mais lentos. Logo, a capacidade da cache é de **64 kB**.

3. What is the size of each cache block?

O tamanho do bloco da cache pode ser determinado ao identificar o stride a partir do qual o tempo de acesso se estabiliza num valor máximo, para arrays maiores que a capacidade da cache (> 64 kB). Este comportamento indica que cada acesso ao array (array[i]) está a mapear para um bloco de cache distinto, resultando numa taxa de *misses* próxima de 100%. Observando o gráfico, notamos que para strides de 16 bytes ou superiores, o tempo de acesso atinge um patamar estável. Isto significa que um passo de 16 bytes já é suficiente para garantir que cada acesso sequencial no loop interno aceda a um novo bloco de cache. Logo, o tamanho de cada bloco da cache é de **16 bytes**.

4. What is the L1 cache miss penalty time?

A penalidade por miss na L1 pode ser estimada subtraindo o tempo de acesso de um hit do tempo de acesso de um miss.

- Tempo de Acesso (Hit): para arrays que cabem na cache (< 64 KiB), a taxa de misses é próxima de 0%. O tempo de acesso médio nestes casos é de aproximadamente 375 ns.
- Tempo de Acesso (Miss): para arrays maiores que a cache e com um stride superior ao tamanho do bloco (≥ 16 bytes), a taxa de misses é próxima de 100%. O tempo de acesso médio nestes casos é de aproximadamente 975 ns.

A diferença entre estes dois valores representa o tempo adicional necessário para resolver um miss na L1, ou seja, a penalidade. Fazendo o cálculo Penalidade por Miss = Tempo (Miss) - Tempo (Hit) = 975 ns - 375 ns = 600 ns, chegamos à conclusão de que o tempo de penalidade por miss na cache L1 é de aproximadamente **600 ns**.

3 Assignment

3.1.1 Modeling the L1 Data Cache

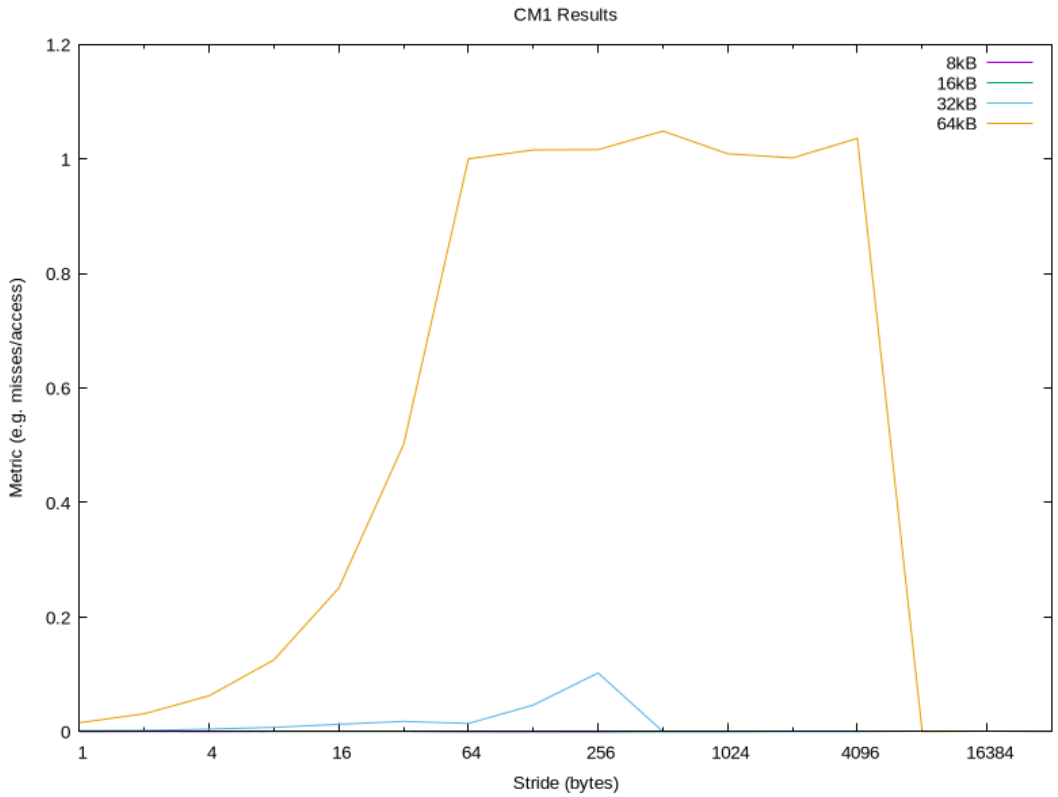
a) What are the processor events that will be analyzed during its execution? Explain their meaning.

Durante a execução do programa, os eventos analisados vão ser falhas da cache L1, que ocorrem quando os dados não são encontrados na cache L1 e é necessário aceder ao próximo nível de memória (cache L2). O programa cm1.c rastreia o evento PAPI_L1_DCM, indicando falhas da cache de dados L1. Esta análise visa quantificar a frequência com que tentamos recuperar dados ausentes da cache L1 durante o tempo de execução do programa.

b) Plot the variation of the average number of misses (*Avg Misses*) with the `stride` size, for each considered dimension of the L1 data cache (8kB, 16kB, 32kB and 64kB).
Note that, you may fill these tables and graphics (as well as the following ones in this report) on your computer and submit the printed version.

Array Size	Stride	Avg Misses	Avg Cycle Time
8kBytes	1	0.000227	0.003035
	2	0.000293	0.003223
	4	0.000128	0.003186
	8	0.000106	0.003063
	16	0.000111	0.003178
	32	0.000107	0.002928
	64	0.000086	0.002592
	128	0.000054	0.002275
	256	0.000030	0.002163
	512	0.000029	0.002220
	1024	0.000024	0.002092
	2048	0.000015	0.002145
	4096	0.000013	0.002189
16kBytes	1	0.000367	0.002492
	2	0.000244	0.002419
	4	0.000245	0.002360
	8	0.000222	0.002282
	16	0.000236	0.002352
	32	0.000216	0.002314
	64	0.000251	0.002319
	128	0.000124	0.002298
	256	0.000043	0.002165
	512	0.000039	0.002162
	1024	0.000027	0.002133
	2048	0.000014	0.002202
	4096	0.000009	0.002289
	8192	0.000009	0.002212

Array Size	Stride	Avg Misses	Avg Cycle Time
32kBytes	1	0.001599	0.002350
	2	0.002144	0.002393
	4	0.004391	0.002343
	8	0.007525	0.002270
	16	0.012820	0.002226
	32	0.017840	0.002360
	64	0.014282	0.002322
	128	0.046490	0.002339
	256	0.102404	0.002311
	512	0.000166	0.002173
	1024	0.000082	0.002113
	2048	0.000040	0.002215
	4096	0.000023	0.002318
	8192	0.000007	0.002332
	16384	0.000008	0.002243
64kBytes	1	0.015664	0.002141
	2	0.031301	0.002030
	4	0.062674	0.002329
	8	0.125373	0.002376
	16	0.250714	0.002381
	32	0.501419	0.002482
	64	1.000155	0.002241
	128	1.015634	0.002119
	256	1.016117	0.002167
	512	1.048536	0.002181
	1024	1.008873	0.002090
	2048	1.001652	0.002418
	4096	1.035758	0.005341
	8192	0.000020	0.002357
	16384	0.000004	0.002287
	32768	0.000005	0.002254



c) By analyzing the obtained results:

- Determine the **size** of the L1 data cache. Justify your answer.

Com base nos dados, é possível concluir que o tamanho da cache L1 de dados é **32 KiB**. Esta conclusão provém de dois fatores principais. O primeiro é o facto de terem sido registados valores baixos até 32 KiB, ou seja, para tamanhos de array de 8 KiB, 16 KiB e 32 KiB, o valor de avg_misses é muito baixo (na ordem de 0.0002 a 0.0016). Isto indica que a maioria dos acessos são hits na cache, o que significa que estes arrays cabem confortavelmente na cache L1. O segundo fator é o facto de existir um salto abrupto em 64 KiB, ou seja, quando o tamanho do array aumenta de 32 KiB para 64 KiB, a avg_misses sofre um aumento significativo, saltando de 0.001599 para 0.015664, um aumento de quase 10 vezes. Este salto acentuado demonstra que um array de 64 KiB já não cabe na cache L1. Portanto, o limite da capacidade da cache L1 está entre 32 KiB e 64 KiB, o que nos leva a concluir que a sua capacidade é de **32 KiB**.

- Determine the **block size** adopted in this cache. Justify your answer.

Com base nos dados, é possível concluir que o tamanho do bloco da cache L1 é de **64 Bytes**. Esta conclusão provém de dois fatores principais. O primeiro é o facto de ocorrer um aumento linear dos misses: Ao analisar os dados para cache_size = 65536, observamos que a avg_misses duplica aproximadamente cada vez que o stride duplica, desde stride 1 até stride 64. Por exemplo, para stride 32, a avg_misses é ~0.5, e para stride 64, sobe para ~1.0. Isto acontece porque, enquanto o stride é menor que o tamanho do bloco, cada novo bloco carregado para a cache ainda serve para múltiplos acessos. O segundo fator é o facto de ocorrer um ponto de saturação, ou seja, no stride 64, a avg_misses atinge um valor próximo de 1.0. Isto significa que quase todos os acessos são um miss. Quando aumentamos o stride para 128, a avg_misses mantém-se em ~1.0 e estabiliza para strides maiores. Este comportamento indica que, a partir de um stride de 64 bytes, cada acesso ao array já está a aceder a um bloco de memória completamente novo. Portanto, o tamanho do bloco da cache tem de ser **64 Bytes**.

- Characterize the **associativity set size** adopted in this cache. Justify your answer.

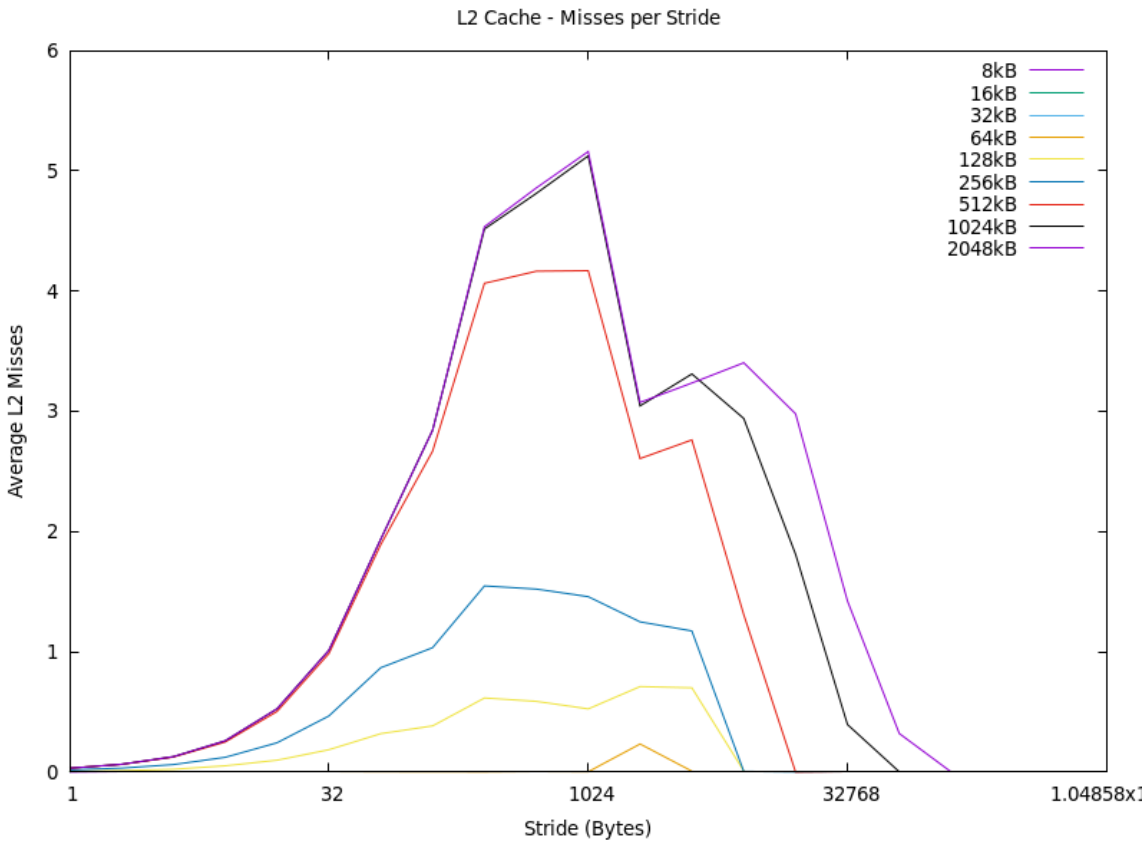
Com base nos dados, é possível concluir que a associatividade da cache L1 de dados é **8**. Para determinar a associatividade usamos a análise de conflict misses. Esta análise foca-se no ponto em que múltiplos acessos mapeiam para o mesmo conjunto da cache, excedendo o número de vias disponíveis e causando um pico de misses. O stride que força todos os acessos para o mesmo conjunto é um múltiplo de (Capacidade / Associatividade). Para uma cache de 32 KiB e uma associatividade de 8, este stride crítico é 32768 / 8 = 4096 bytes. Nos dados experimentais para cache_size de 32768, observamos que para uma stride de 4096, o número de avg_misses cai para um valor próximo de zero (0.000025). Com este stride, o loop de teste realiza 32768 / 4096 = 8 acessos. Como estes 8 acessos não causam um pico de misses, significa que a cache consegue armazenar 8 blocos diferentes no mesmo conjunto sem conflitos. Isto demonstra que a associatividade da cache é pelo menos 8. Dado que a associatividade é tipicamente uma potência de 2, podemos concluir que a associatividade é **8**. Se fosse 4, teríamos observado um pico de misses com este padrão de acesso, o que não aconteceu.

3.1.2 Modeling the L2 Cache

a) Describe and justify the changes introduced in this program.

De modo a analisar as características da cache L2, modificámos o código do programa cm1.c utilizado no exercício anterior. As mudanças efetuadas foram duas. A primeira foi a substituição de PAPI_L1_DCM para PAPI_L2_DCM, de modo a ser possível analisar as falhas da cache de dados L2. A segunda mudança foi um aumento da variável CACHE_MAX, de 64 KB para 2048 KB, tendo em conta as maiores dimensões da cache L2.

b) Plot the variation of the average number of misses (*Avg Misses*) with the `stride` size, for each considered dimension of the L2 cache.



c) By analyzing the obtained results:

- Determine the **size** of the L2 cache. Justify your answer.

Com base nos dados, é possível concluir que o tamanho da cache L2 de dados é **256 kB**. Ao olhar para o gráfico, é possível verificar que até este valor, a taxa média de *misses* mantém-se baixa e estável, ou seja, os arrays ainda cabem na cache. No entanto, ao passar para 512 kB, observa-se um salto claro no número de misses, sinal de que a cache L2 deixa de acomodar o array e parte dos acessos é redirecionada para níveis superiores de memória, mais lentos. A partir daí, a taxa estabiliza, refletindo o novo padrão de acesso. Assim, o ponto de rutura entre 256 kB e 512 kB confirma que a capacidade da cache L2 é de **256 kB**.

- Determine the **block size** adopted in this cache. Justify your answer.

Com base nos dados, é possível concluir que o tamanho do bloco da cache L2 é de **64 Bytes**. Esta conclusão provém de dois fatores principais. O primeiro é o facto de ocorrer um aumento da taxa de misses, ou seja, ao analisar os dados para um cache_size de 512 kB, observamos que a avg_misses escala quase linearmente com o stride até um passo de 64 bytes. Por exemplo, para stride 32, a avg_misses é de 0.985, e para stride 64, sobe para 1.890 (aproximadamente o dobro). Este comportamento é esperado enquanto múltiplos acessos ainda caem dentro do mesmo bloco de cache. O segundo é o facto de ocorrer um ponto de saturação, ou seja, o aumento da taxa de misses abranda significativamente quando o stride passa de 64 para 128 bytes. Este ponto de saturação indica que, a partir de um stride de 64 bytes, cada acesso ao array já está a forçar o carregamento de um novo bloco de memória. Se o bloco fosse maior, por exemplo 128 bytes, o salto na taxa de misses só iria ocorrer para strides superiores a 128. Logo, o ponto de viragem no comportamento da taxa de misses em stride **64 Bytes** revela que este é o tamanho do bloco da cache L2.

- Characterize the **associativity set size** adopted in this cache. Justify your answer.

Com base nos dados, é possível concluir que a associatividade da cache L2 é **8**. A análise de associatividade foca-se em conflict misses para um cache_size igual à capacidade da cache, que neste caso é de 256 kB. O stride crítico que força todos os acessos a mapearem para o mesmo conjunto da cache é um múltiplo de (Capacidade / Associatividade). Para uma cache de 256 kB e uma associatividade de 8, este stride é 262144 / 8 = 32768 bytes. Nos dados experimentais para cache_size 262144, observamos que para stride 32768, a avg_misses é extremamente baixa (0.000118). Com este stride, o loop de teste realiza 262144 / 32768 = 8 acessos distintos. Como estes 8 acessos não causam um pico de misses, significa que a cache L2 consegue armazenar 8 blocos de memória diferentes no mesmo conjunto sem gerar conflitos. Isto demonstra que a associatividade da cache é pelo menos 8. Sabendo que a associatividade é tipicamente uma potência de 2, podemos concluir que a associatividade é **8**.

3.2 Profiling and Optimizing Data Cache Accesses

3.2.1 Straightforward implementation

a) What is the total amount of memory that is required to accommodate each of these matrices?

Cada matriz tem dimensão 512 × 512, resultando em 262144 elementos. Como o tipo usado é int16_t, cada elemento ocupa 2 bytes. Assim, a memória total necessária por matriz é 262144 × 2 = 524288 bytes, ou seja, 512 kB (0,5 MB). Portanto, cada uma das matrizes mul1, mul2 e res requer **512 kB** de memória.

b) Fill the following table with the obtained data.

Total number of L1 data cache misses	*135.252931* $\times 10^6$
Total number of load / store instructions completed	*536.871896* $\times 10^6$
Total number of clock cycles	*734.696436* $\times 10^6$
Elapsed time	*0.215579* seconds

c) Evaluate the resulting L1 data cache *Hit-Rate*:

$$HitRate = 1 - MissRate = 1 - \frac{Misses}{Accesses} = 1 - \frac{135.252931 * 10^6}{536.871896 * 10^6} \approx 0.74808$$

Logo, é possível concluir que a taxa de acertos é aproximadamente **74.81%.**

3.2.2 First Optimization: Matrix transpose before multiplication [2]

a) Fill the following table with the obtained data.

Total number of L1 data cache misses	*4.219293* $\times 10^6$
Total number of load / store instructions completed	*536.871896* $\times 10^6$
Total number of clock cycles	*665.864738* $\times 10^6$
Elapsed time	*0.195379* seconds

b) Evaluate the resulting L1 data cache *Hit-Rate*:

$$HitRate = 1 - MissRate = 1 - \frac{Misses}{Accesses} = 1 - \frac{4.219293 * 10^6}{536.871896 * 10^6} \approx 0.99214$$

Logo, é possível concluir que a taxa de acertos é aproximadamente **99.21%.**

c) Fill the following table with the obtained data.

Total number of L1 data cache misses	*4.355812* $\times 10^6$
Total number of load / store instructions completed	*537.396188* $\times 10^6$
Total number of clock cycles	*663.482794* $\times 10^6$
Elapsed time	*0.194681* seconds

Comment on the obtained results when including the matrix transposition in the execution time:

$$HitRate = 1 - MissRate = 1 - \frac{Misses}{Accesses} = 1 - \frac{4.355812 * 10^6}{537.396188 * 10^6} \approx 0.99189$$

Ao incluir a transposição da matriz na medição, verificou-se um pequeno aumento no número de instruções de load/store e nos L1 cache misses, resultado das leituras e escritas adicionais. No entanto, o impacto no desempenho foi mínimo, uma vez que o tempo médio manteve-se em ~0,195s e os ciclos até reduziram ligeiramente, dentro da variação normal do sistema. Assim, o custo da transposição é insignificante face ao ganho obtido.

d) Compare the obtained results with those that were obtained for the straightforward implementation, by calculating the difference of the resulting hit-rates (ΔHitRate) and the obtained speedups.

$\Delta HitRate = HitRate_{mm2} - HitRate_{mm1}$: $0.99189 - 0.74808 = \mathbf{0.24381}$
$Speedup(\#Clocks) = \#Clocks_{mm1} / \#Clocks_{mm2}$: $\frac{734.696436}{663.482794} = \mathbf{1.107333}$
$Speedup(Time) = Time_{mm1} / Time_{mm2}$: $\frac{0.215579}{0.194681} = \mathbf{1.107345}$
Comment: A otimização por transposição de matriz resultou numa melhoria de desempenho significativa em comparação com a implementação original. O principal fator para esta melhoria foi o aumento drástico da taxa de acertos na cache L1 de dados, que subiu de **74.81%** para **99.19%** (ganho de **24.38 pontos percentuais**). Esta melhoria deve-se à otimização da localidade espacial dos acessos à matriz mul2. Na versão original, os acessos eram feitos por colunas, resultando em saltos grandes na memória e, consequentemente, num elevado número de *cache misses*. Na versão otimizada, os acessos a ambas as matrizes são sequenciais, o que permite que o hardware explore eficientemente a cache. O tempo de execução e o número de ciclos de relógio também diminuíram. O *speedup* obtido foi de aproximadamente **1.11x**, tanto em tempo de execução como em ciclos de relógio, o que confirma que a otimização foi bem-sucedida e que o custo da transposição da matriz foi largamente compensado pelos ganhos de eficiência no acesso à memória.

3.2.3 Second Optimization: Blocked (tiled) matrix multiply [2]

a) How many matrix elements can be accommodated in each cache line?

Tendo em conta que determinámos anteriormente que o tamanho da linha de cache é 64 bytes e que os elementos da matriz são do tipo int16_t, que ocupam 2 bytes cada, logo, o número de elementos que cabem numa única linha de cache é 64 bytes / 2 bytes = **32**.

b) Fill the following table with the obtained data.

Total number of L1 data cache misses	*3.253596* $\times 10^6$
Total number of load / store instructions completed	*537.547779* $\times 10^6$
Total number of clock cycles	*284.715616* $\times 10^6$
Elapsed time	*0.083541* seconds

c) Evaluate the resulting L1 data cache *Hit-Rate*:

$$HitRate = 1 - MissRate = 1 - \frac{Misses}{Accesses} = 1 - \frac{3.253596 * 10^6}{537.547779 * 10^6} \approx 0.99394$$

Logo, é possível concluir que a taxa de acertos é aproximadamente **99.39%.**

d) Compare the obtained results with those that were obtained for the straightforward implementation, by calculating the difference of the resulting hit-rates (ΔHitRate) and the obtained speedup.

$\Delta HitRate = HitRate_{mm3} - HitRate_{mm1}$: $0.99394 - 0.74808 = \mathbf{0.24586}$
$Speedup(\#Clocks) = \#Clocks_{mm1}/\#Clocks_{mm3}$: $\frac{734.696436}{284.715616} = \mathbf{2.580457}$
Comment: A otimização por blocos obteve um desempenho drasticamente superior ao da implementação original. O *speedup* alcançado foi de aproximadamente **2.58x**. A principal causa para este ganho de desempenho foi a melhoria excecional na utilização da cache L1. A taxa de acertos aumentou em **24.59 pontos percentuais**, passando de 74.81% para **99.39%**. Esta melhoria deve-se à excelente localidade de dados que a técnica de *cache blocking* proporciona. Ao dividir a matriz em blocos que cabem na cache, o algoritmo reutiliza intensivamente os dados que já estão nos níveis mais rápidos da memória, reduzindo o número de L1 data cache misses.

e) Compare the obtained results with those that were obtained for the matrix transpose implementation by calculating the difference of the resulting hit-rates (ΔHitRate) and the obtained speedup. If the obtained speedup is positive, but the difference of the resulting hit-rates is negative, how do you explain the performance improvement? (Hint: study the hit-rates of the L2 cache for both implementations;)

$\Delta\text{HitRate} = \text{HitRate}_{mm3} - \text{HitRate}_{mm2}$: $0.99394 - 0.99189 = \mathbf{0.00205}$
$\text{Speedup(\#Clocks)} = \#\text{Clocks}_{mm2}/\#\text{Clocks}_{mm3}$: $\frac{663.482794}{284.715616} = \mathbf{2.330335}$
Comment: A implementação com *cache blocking* demonstrou ser **2.33x mais rápida** que a implementação com transposição de matriz. Curiosamente, esta melhoria de desempenho não se deve a um ganho significativo na taxa de acertos da cache L1, que aumentou apenas **0.21 pontos percentuais**. Ambas as otimizações já são quase perfeitas a nível da L1. A verdadeira razão para o speedup reside na utilização muito mais eficiente da cache L2. A técnica de *blocking* maximiza a localidade temporal, processando a multiplicação em pequenos blocos que cabem inteiramente na hierarquia de cache. Isto permite que os dados sejam reutilizados intensivamente a partir da cache L2, minimizando a necessidade de aceder a níveis de memória mais lentos. Em contraste, a técnica de transposição, embora melhore a localidade espacial, ainda força a leitura repetida de grandes volumes de dados da memória para a L2, resultando numa taxa de *misses* na L2 muito superior.

3.2.3 Comparing results against the CPU specifications

Now that you have characterized the cache on your lab computer, you are going to compare it against the manufacturer's specification. For this you can check the device's datasheet, or make use of the command `lscpu`. Comment on the results.

A comparação entre os resultados experimentais e as especificações do CPU obtidas via **lscpu** mostra uma correspondência quase perfeita, validando a metodologia usada. A cache L1 de dados revelou capacidade de **32 kB**, exatamente o valor indicado pelo fabricante, confirmado pelo aumento súbito da taxa de misses ao ultrapassar esse tamanho. Para a L2, os testes apontaram **256 kB**, também em conformidade com as especificações, com o ponto de inflexão claro quando os arrays passaram desse limite. Já o tamanho de bloco foi identificado como **64 bytes**, valor padrão da arquitetura, evidenciado pelo aumento de misses a partir de strides desse tamanho.