# Second Lab Assignment:
# System Modeling and Profiling

## 2. Exercise

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 | 8 kBytes | 16 kBytes | 32 kBytes | 64 kBytes | 128 kBytes | 256 kBytes |
|---|---|---|---|---|---|---|
| t2-t1 | 0.001711 | 0.003440 | 0.007006 | 0.017020 | 0.042189 | 0.094689 |
| # accesses a[i] | 10649600 | 22937600 | 49152000 | 104857600 | 222822400 | 471859200 |
| # mean access time | 0.16 ns | 0.15 ns | 0.14 ns | 0.16 ns | 0.19 ns | 0.20 ns |

Computador utilizado: **lab7p2**
Analisando o output do programa fornecido, podemos reparar que existe um grande aumento no tempo de acesso à memória quando o tamanho do array passa de 32 kBytes para 64 kBytes, o que nos leva à conclusão de que a cache tem um tamanho de 32 kBytes.

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?

Pela análise do gráfico, é possível repararmos que o tempo de acesso cresce drasticamente quando passamos de um array de 64 kBytes para um de 128 kBytes, sugerindo que, para $size > 64\ kBytes$, o array deixa de caber em cache, o que nos leva a concluir que a cache é de 64 kBytes.

3. What is the size of each cache block?

Analisando agora o valor do *stride*, podemos notar que, para $stride \geq 16$ (quando $array\ size > 64\ kBytes$ ), os tempos de acesso passam a ser relativamente constantes, o que nos leva a concluir que cada bloco da cache tem um tamanho aproximado de:

- $Block\ size = 16 \times sizeof(array[i])$;
- onde $array[i]$ representa uma entrada do array.

4. What is the L1 cache miss penalty time?

Tendo em conta que o tamanho da cache é de 64 kBytes e que o tamanho de cada bloco é determinado quando $stride = 16$, utilizando os tempos de acesso para o array de 64 kBytes e para o de 128 kBytes, podemos aproximar o miss penalty time a:

- $Miss\ penalty \simeq 950 - 350 = 600\ ns$.

# 3. Procedure

## 3.1.1 Modeling the L1 Data Cache

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

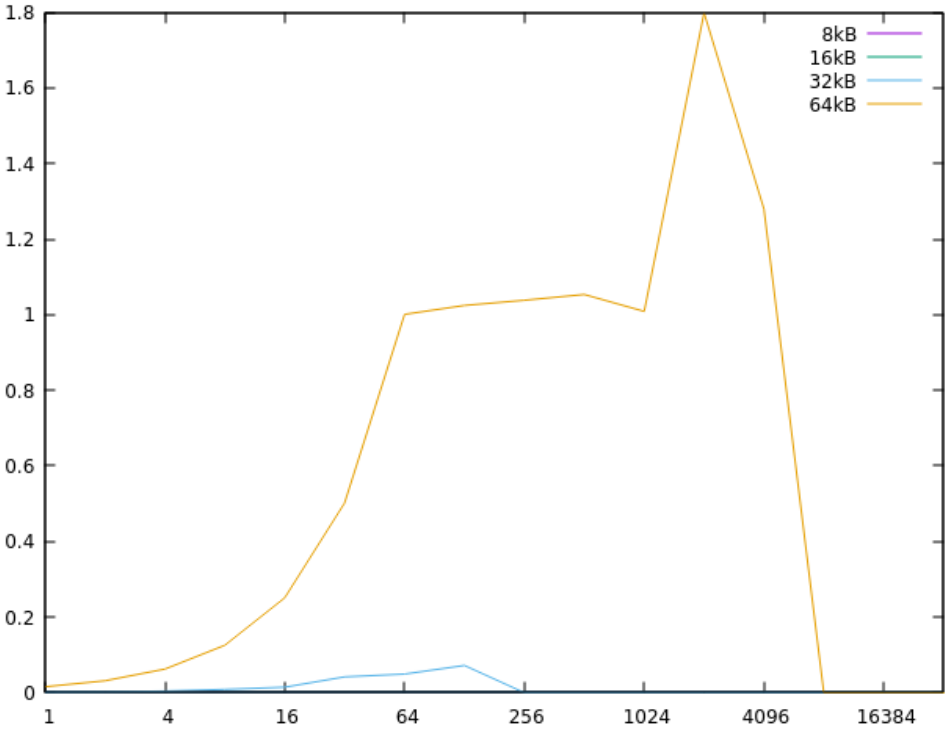
**PAPI_L1_DCM:**

- Evento que lida com as *data cache misses* da L1.

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 Cycl Time |
|---|---|---|---|
| 8kBytes | 1 | 0.000166 | 0.002313 |
| | 2 | 0.000088 | 0.002248 |
| | 4 | 0.000027 | 0.002252 |
| | 8 | 0.000053 | 0.002204 |
| | 16 | 0.000029 | 0.002231 |
| | 32 | 0.000032 | 0.002203 |
| | 64 | 0.000026 | 0.002202 |
| | 128 | 0.000012 | 0.002056 |
| | 256 | 0.000004 | 0.002001 |
| | 512 | 0.000004 | 0.001984 |
| | 1024 | 0.000008 | 0.001960 |
| | 2048 | 0.000002 | 0.002042 |
| | 4096 | 0.000002 | 0.002076 |
| 16kBytes | 1 | 0.000094 | 0.002245 |
| | 2 | 0.000075 | 0.002242 |
| | 4 | 0.000111 | 0.002246 |
| | 8 | 0.000063 | 0.002126 |
| | 16 | 0.000062 | 0.002248 |
| | 32 | 0.000059 | 0.002249 |
| | 64 | 0.000046 | 0.002195 |
| | 128 | 0.000031 | 0.002163 |
| | 256 | 0.000013 | 0.002042 |
| | 512 | 0.000006 | 0.001961 |
| | 1024 | 0.000001 | 0.001934 |
| | 2048 | 0.000002 | 0.002041 |
| | 4096 | 0.000001 | 0.002130 |
| | 8102 | 0.000001 | 0.002068 |

| Array Size | Stride | Avg Misses | Avg Cycl Time |
|---|---|---|---|
| 32kBytes | 1 | 0.001122 | 0.002204 |
| | 2 | 0.001898 | 0.002184 |
| | 4 | 0.004055 | 0.002187 |
| | 8 | 0.008755 | 0.002140 |
| | 16 | 0.014306 | 0.002132 |
| | 32 | 0.041683 | 0.002067 |
| | 64 | 0.049171 | 0.002252 |
| | 128 | 0.071881 | 0.002220 |
| | 256 | 0.000101 | 0.002108 |
| | 512 | 0.000039 | 0.002000 |
| | 1024 | 0.000018 | 0.001953 |
| | 2048 | 0.000004 | 0.002006 |
| | 4096 | 0.000002 | 0.002161 |
| | 8102 | 0.000001 | 0.002138 |
| | 16384 | 0.000000 | 0.002062 |
| 64kBytes | 1 | 0.015651 | 0.001941 |
| | 2 | 0.031304 | 0.001876 |
| | 4 | 0.062663 | 0.002135 |
| | 8 | 0.125346 | 0.002175 |
| | 16 | 0.250727 | 0.002228 |
| | 32 | 0.501469 | 0.002105 |
| | 64 | 1.000922 | 0.001898 |
| | 128 | 1.024413 | 0.001989 |
| | 256 | 1.037951 | 0.001840 |
| | 512 | 1.053551 | 0.001881 |
| | 1024 | 1.008622 | 0.001832 |
| | 2048 | 1.798345 | 0.004762 |
| | 4096 | 1.279631 | 0.007525 |
| | 8102 | 0.000003 | 0.002153 |
| | 16384 | 0.000001 | 0.002146 |
| | 32768 | 0.000000 | 0.002036 |



c) By analyzing the obtained results:

• Determine the size of the L1 data cache. Justify your answer.

Pela análise dos valores obtidos, e consequentemente do gráfico, podemos reparar que, nos arrays de 8, 16 e 32 kBytes, o número de average-misses mantém-se relativamente constante (muito perto de zero).

Contudo, é possível notar a existência de uma grande discrepância entre estes valores quando o tamanho do array passa para 64 kBytes (miss-rate aumenta bastante), justificado pelo facto de, a partir desse tamanho, o array deixar de caber na cache L1. Desta forma, concluímos que o tamanho da cache é de 32 kBytes.

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

Analisando tanto o gráfico como a tabela, podemos reparar que, até $stride = 64$, o número de *average misses* aumenta mas mantém-se bastante baixo, o que nos indica que vários elementos do array cabem no mesmo bloco da cache.

No entanto, para $stride \geq 64$, reparamos que as *average misses* deixam de ter um crescimento tão notável, tornando-se relativamente constantes (facilmente percetível no gráfico, no array de 64 kBytes). Percebemos, então, que o *stride* ultrapassa aí o tamanho de cada bloco, estando, por isso, sempre a aceder a diferentes blocos e, consequentemente, resultando em *average-misses* semelhantes. Logo, cada bloco da cache consegue armazenar cerca de 64 posições do array (do tipo $uint8\_t$, no caso dos testes), de onde concluímos que:

- $Block\ size = 64 \times sizeof(uint8\_t) = 64 \times 1 = 64\ Bytes$;

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

Para determinar a associatividade da cache, basta analisar o gráfico quando o tamanho testado é maior do que o da L1, garantindo que cada acesso seja mapeado num set diferente. Neste caso, temos apenas como referência $cache\ size = 64\ kB$, visto ser o único maior que os 32 kBytes definidos anteriormente.

Observemos apenas os resultados com $cache\ size = 64\ kB$:

- a partir de $stride = 2048$, reparamos logo numa diminuição drástica do número de *average misses*;
- chegando a $stride = 8192$, o número de *average misses* passa a ser sempre $\approx 0$.

Esta segunda observação ajuda-nos a chegar a uma conclusão sobre o número de vias da cache, pois tendo em conta que $stride = 8192 = ⅛\ cache\ size$, e que praticamente deixam de existir misses a partir deste valor, podemos concluir que quaisquer acessos feitos com $stride \geq 8192$ serão mapeados nos mesmos sets da cache, os quais possuem vias suficientes para os armazenar, conseguindo assim, um número de *average misses* $\approx 0$.

Neste sentido, valores do array resultantes de $stride \in [0, \frac{1}{8}n, \frac{2}{8}n, ..., \frac{7}{8}n]$, onde $n = cache\ size = 64\ kB$, serão então sempre mapeados nos mesmos sets de associatividade da cache, o que nos leva a deduzir que a cache tem portanto 8 vias de associatividade.

### 3.1.2 Modeling the L2 Cache

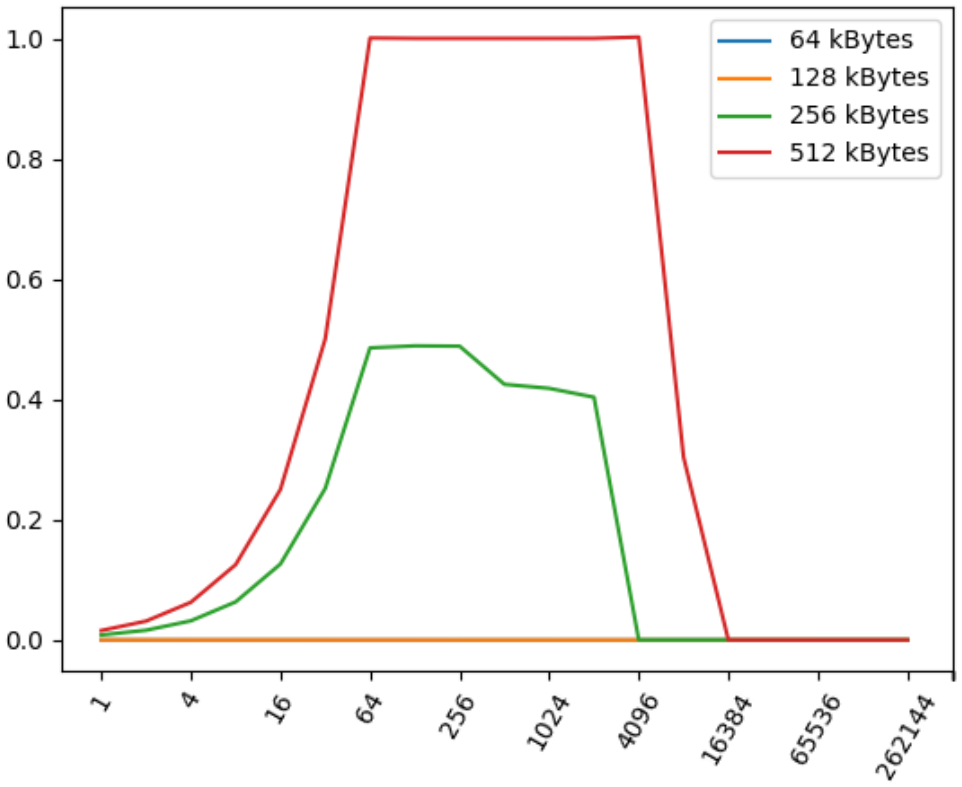
a) Describe and justify the changes introduced in this program.

Foram feitas duas mudanças ao programa:

- substituímos o evento *PAPI_L1_DCM* no event set do processador pelo evento *PAPI_L2_DCM*, para, assim, conseguirmos analisar o número de misses na cache L2;

- alterámos o tamanho mínimo e máximo da cache (*CACHE_MIN e CACHE_MAX*), de modo a forçarmos a utilização da cache L2 (não garantido com tamanhos menores do array), de 8 kBytes e 64 kBytes para 64 kBytes e 512 kBytes, respectivamente.

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.

Analisando o novo gráfico, podemos desde logo verificar um enorme incremento no número de *average misses* quando passamos de um array de 256 kBytes para um de 512 kBytes, sugerindo que a cache deixa de ter espaço suficiente para guardar arrays de tamanho $\geq$ 512 kBytes (daí gerar tantos *misses*). Podemos, então, concluir que a cache L2 tem um tamanho de 256 kBytes.

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

Pela análise do gráfico, podemos reparar que, até $stride = 64$, o número de *average misses* aumenta, o que nos indica que vários elementos do array cabem no mesmo bloco da cache.

No entanto, para $stride \geq 64$, reparamos que as *average misses* deixam de ter um crescimento tão notável, tornando-se relativamente constantes (perceptível no gráfico). Percebemos, então, que o *stride* ultrapassa aí o tamanho de cada bloco, estando, por isso, sempre a aceder a diferentes blocos e, consequentemente, resultando em *average-misses* semelhantes. Logo, cada bloco da cache consegue armazenar cerca de 64 posições do array (do tipo $uint8\_t$, no caso dos testes), de onde concluímos que:

- $Block\ size = 64 \times sizeof(uint8\_t) = 64 \times 1 = 64\ Bytes$ ;

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

Para determinar a associatividade da cache, analisamos o gráfico quando o tamanho testado é maior do que o da L2, garantindo que cada acesso seja mapeado num set diferente. Neste caso, temos apenas como referência $cache\ size = 512\ kB$, visto ser o único maior que os 256 kBytes definidos anteriormente.

Observemos apenas os resultados com $cache\ size = 512\ kB$:

- a partir de $stride = 4096$, reparamos logo numa diminuição drástica do número de *average misses*;
- chegando a $stride = 16384$, o número de *average misses* passa a ser sempre $\approx 0$.

Esta segunda observação ajuda-nos a chegar a uma conclusão sobre o número de vias da cache, pois tendo em conta que $stride = 16384 = \frac{1}{32}\ cache\ size$, e que praticamente deixam de existir misses a partir deste valor, podemos concluir que quaisquer acessos feitos com $stride \geq 16384$ serão mapeados nos mesmos sets da cache, os quais possuem vias suficientes para os armazenar, conseguindo assim um número de *average misses* $\approx 0$.

Neste sentido, valores do array resultantes de $stride \in [0, \frac{1}{32}n, \frac{2}{32}n, ..., \frac{31}{32}n]$, onde $n = cache\ size = 512\ kB$, serão então sempre mapeados nos mesmos sets de associatividade da cache, o que nos leva a deduzir que a cache tem portanto 32 vias de associatividade (isto não faz sentido nenhum, mas oh well o gráfico dá isso).

## 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?

Dados do programa:
- matrizes $N \times N$, onde $N = 512$;
- entradas das matrizes do tipo $int16\_t$.

Logo, podemos concluir que:
$Total_{memory} = 512 \times 512 \times sizeof(int16\_t) = 262144 \times 2\ Bytes = 524288\ Bytes$
$= \frac{524288}{2^{10}}\ kBytes = 512\ kBytes$ ❒

b) Fill the following table with the obtained data.

| | |
|---|---|
| Total number of L1 data cache misses | $134.851452 \times 10^6$ |
| Total number of load / store instructions completed | $536.881215 \times 10^6$ |
| Total number of clock cycles | $817.327547 \times 10^6$ |
| Elapsed time | 0.240938 seconds |

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

$$Hit\ Rate = 1 - Miss\ Rate = 1 - \frac{\#L1\ data\ cache\ misses}{\#load/store\ instructions} = 1 - \frac{134.851452 \times 10^6}{536.881215 \times 10^6} =$$
$$\simeq 1 - 0.25118 = 0.74882 = 74.882\ \%$$ ❒

### 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 | $4.212240 \times 10^6$ |
| Total number of load / store instructions completed | $536.880349 \times 10^6$ |
| Total number of clock cycles | $744.649013 \times 10^6$ |
| Elapsed time | 0.219514 seconds |

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

$$Hit\ Rate = 1 - Miss\ Rate = 1 - \frac{\#L1\ data\ cache\ misses}{\#load/store\ instructions} = 1 - \frac{4.212240 \times 10^6}{536.880349 \times 10^6} =$$

$$\simeq 1 - 0.00785 = 0.99215 = 99.215\ \%\ \square$$

Existe um aumento significativo no Hit-Rate quando se usa a transposta da matriz m2 para a multiplicação de matrizes. Isto ocorre porque, ao calcular a transposta de uma matriz e multiplicá-la com outra matriz, não necessitamos de avançar uma linha a cada loop interno (no algoritmo de multiplicação de matrizes), ou seja, evitamos sucessivos acessos a posições de memória longínquas.

c) Fill the following table with the obtained data.

| | |
|---|---|
| Total number of L1 data cache misses | $4.484687 \times 10^6$ |
| Total number of load / store instructions completed | $537.405504 \times 10^6$ |
| Total number of clock cycles | $739.563875 \times 10^6$ |
| Elapsed time | 0.218015 seconds |

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

Quando é incluída a transposição das matrizes no tempo de execução do programa, as diferenças nos parâmetros analisados são escassas, pois o número de instruções realizadas não aumenta significativamente. No entanto, registam-se mais L1 data cache misses e mais load/store instructions complete.

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.

| |
|---|
| $\Delta\text{HitRate} = \text{HitRate}_{\text{mm2}} - \text{HitRate}_{\text{mm1}}$: $0.99215 - 0.74882 = 0.24333$ |
| $\text{Speedup}(\#\text{Clocks}) = \#\text{Clocks}_{\text{mm1}}/\#\text{Clocks}_{\text{mm2}}$: $\frac{817.327547 \times 10^6}{744.649013 \times 10^6} \simeq 1.0096$ |
| $\text{Speedup}(\text{Time}) = \text{Time}_{\text{mm1}}/\text{Time}_{\text{mm2}}$: $\frac{0.240938}{0.219514} \simeq 1.0096$ |
| Comment:<br>Como era de esperar, em teoria, todos os parâmetros analisados melhoram em comparação com a implementação que não envolve a transposição de uma das matrizes. O speedup é notável, quer em número de ciclos de relógio, quer em tempo de execução (sendo igual proporcionalmente). Para além disso, os misses da cache L1 são muito mais raros quando usamos esta 2ª implementação, devido à maior proximidade dos acessos à memória, aproveitando assim a proximidade local (localidade espacial), quando calculamos primeiro a transposta da matriz a multiplicar. |

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

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

$$\#matrix_{elements} = \frac{line\ size}{sizeof(int16\_t)} = \frac{64\ Bytes}{2\ Bytes} = 32\ elementos\ \square$$

b) Fill the following table with the obtained data.

| | |
|---|---|
| Total number of L1 data cache misses | $5.867024 \times 10^6$ |
| Total number of load / store instructions completed | $537.025882 \times 10^6$ |
| Total number of clock cycles | $396.506520 \times 10^6$ |
| Elapsed time | 0.116885 seconds |

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

$$Hit\ Rate = 1 - Miss\ Rate = 1 - \frac{\#L1\ data\ cache\ misses}{\#load/store\ instructions} = 1 - \frac{5.867024 \times 10^6}{537.025882 \times 10^6} =$$
$$\simeq 1 - 0.01093 = 0.98907 = 98.907\ \%\ \square$$

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.

| |
|---|
| $\Delta\text{HitRate} = \text{HitRate}_{\text{mm3}} - \text{HitRate}_{\text{mm1}}$: $0.98907 - 0.74882 = 0.24025$ |
| $\text{Speedup}(\#\text{Clocks}) = \#\text{Clocks}_{\text{mm1}} / \#\text{Clocks}_{\text{mm3}}$: $\frac{817.327547 \times 10^6}{396.506520 \times 10^6} \simeq 2.0613$ |
| Comment:<br>É evidente pela análise dos resultados que esta implementação beneficia o algoritmo.<br>Nesta implementação, tomamos partido da localidade espacial da cache, visto que processamos as matrizes, e por sua vez a sua multiplicação, em blocos, aproveitando assim o tamanho de cada bloco da cache por completo, levando a uma pequena melhoria no Hit-Rate e a um speedup bastante bom a nível de clock cycles (2.0613). |

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;)

| |
|---|
| $\Delta\text{HitRate} = \text{HitRate}_{\text{mm3}} - \text{HitRate}_{\text{mm2}}$: $0.98907 - 0.99215 = -0.00308$ |
| $\text{Speedup}(\#\text{Clocks}) = \#\text{Clocks}_{\text{mm2}} / \#\text{Clocks}_{\text{mm3}}$: $\frac{744.649013 \times 10^6}{396.506520 \times 10^6} \simeq 1.8780$ |
| Comment:<br>Em termos de Hit-Rate, esta nova implementação é um pouco pior, mas a diferença continua a ser insignificante, pois, mesmo com blocking o principal problema mantém-se, as matrizes não cabem em cache, originando por isso um número de capacity misses semelhantes. Contudo, é de notar o Speedup relativamente grande ( 1.8780), o que se deve ao facto de nesta nova implementação não ser necessário haver a transposição de uma das matrizes (uma operação com um custo bastante elevado). |

### 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.

| | As nossas conclusões | Especificação da Máquina [4 CPUs] |
|---|---|---|
| - L1 cache size: | $32\ KB$ | $\frac{128\ KiB}{4\ CPUs} = 32\ kBytes$ |
| - L1 associativity set size: | 8-ways | 8-ways |
| - L2 cache size: | $256\ KB$ | $\frac{1\ MiB}{4\ CPUs} = \frac{1024\ KiB}{4\ CPUs} = 256\ kBytes$ |
| - L2 associativity set size: | 32-ways | 8-ways |

Comparando estes valores, notamos logo uma semelhança entre as especificações da máquina e os valores que obtivemos e analisámos, dando a entender que se aproximam bastante da realidade.

No entanto, há apenas uma característica que se afasta do seu valor real - o número de vias de associatividade da cache L2. Em vez de um set com 8 vias (valor mais ou menos esperado), os gráficos obtidos apontam para 32 vias, o que, de facto, nos surpreendeu por não fazer muito sentido. Aspetos como o estado do computador no momento em que o programa correu, ou talvez a L1 influenciar de alguma forma os valores observados na L2, podem ser uma possível justificação para tal.

```
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   36 bits physical, 48 bits virtual
CPU(s):                          4
On-line CPU(s) list:             0-3
Thread(s) per core:              1
Core(s) per socket:              4
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           58
Model name:                      Intel(R) Core(TM) i5-3570 CPU @ 3.40GHz
Stepping:                        9
CPU MHz:                         1600.000
CPU max MHz:                     3800.0000
CPU min MHz:                     1600.0000
BogoMIPS:                        6784.85
Virtualization:                  VT-x
L1d cache:                       128 KiB
L1i cache:                       128 KiB
L2 cache:                        1 MiB
L3 cache:                        6 MiB
```

```
NAME ONE-SIZE ALL-SIZE WAYS TYPE        LEVEL
L1d       32K     128K    8 Data            1
L1i       32K     128K    8 Instruction     1
L2       256K       1M    8 Unified         2
L3         6M       6M   12 Unified         3
```