Unioeste - Universidade Estadual do Oeste do Paraná
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

Memória CACHE – Aula Prática P03

Luiz Eduardo Garzon de Oliveira

Cascavel, 06 de janeiro de 2023

Objetivo: Implementar um algoritmo que multiplica 2 matrizes que permita a demonstração do efeito positivo ou negativo da memória CACHE com relação a forma de acesso aos dados de um programa.

Etapa 01: Implementar o algoritmo clássico para multiplicação entre duas matrizes (método original e método transposta). Código cache.c arquivado junto a pasta OAC P03 CACHE.

- Tipo de dados Double
- Matrizes alocadas dinamicamente
- Preenchimento das matrizes com valores aleatórios
- Condição de multiplicação c1 == l2
- Função para criar matriz
- Função para preencher matriz de forma aleatória
- Função para multiplicar matriz método original
- Função para transpor matriz
- Função para multiplicar matriz transposta
- Função para printar matriz (para testes)
- Biblioteca C time.h para mensurar tempo de execução

Características da Máquina:

Processador: AMD Ryzen 3 3250U

Freqüência do processador original:	**2600.0 MHz**
Número de núcleos de CPU:	**2**
L1D :	**64 KBytes**
L1i :	**128 Kbytes**
L2:	**1 MBytes**
L3:	**4 MBytes**

Etapa 02: Validação dos Resultados

Site Matrix Calculator para comparar os resultados gerados pelo algoritmo, alguns testes feitos manualmente de forma escrita.

Valores utilizados para testes: números de 0 a 10 (double) para Matrizes A e B

```
for (int i = 0; i < l; i++)
{
    for (int j = 0; j < c; j++)
    {
        matriz[i][j] = (double) (rand() % 10);
        // matriz[i][j] = ((double)(rand() % RAND_MAX)) / ((double)(rand() % RAND_MAX));
    }
}
return matriz;
```

2x2:

Terminal Original

```
2.000000 4.000000
2.000000 3.000000

0.000000 5.000000
4.000000 0.000000

16.000000 10.000000
12.000000 10.000000
```

Matrix Calculator

$$\begin{pmatrix} 2 & 4 \\ 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 0 & 5 \\ 4 & 0 \end{pmatrix} = \begin{pmatrix} 16 & 10 \\ 12 & 10 \end{pmatrix}$$

Terminal Transposta

```
9.000000 3.000000
6.000000 6.000000

6.000000 5.000000
3.000000 6.000000

6.000000 3.000000
5.000000 6.000000

63.000000 63.000000
54.000000 66.000000
```

Matrix Calculator

$$\begin{pmatrix} 9 & 3 \\ 6 & 6 \end{pmatrix} \cdot \begin{pmatrix} 6 & 5 \\ 3 & 6 \end{pmatrix} = \begin{pmatrix} 63 & 63 \\ 54 & 66 \end{pmatrix}$$

3x3:

Terminal Original

```
5.000000 4.000000 3.000000
7.000000 1.000000 3.000000
9.000000 4.000000 1.000000

2.000000 6.000000 6.000000
4.000000 6.000000 5.000000
6.000000 7.000000 7.000000

44.000000 75.000000 71.000000
36.000000 69.000000 68.000000
40.000000 85.000000 81.000000
```

Matrix Calculator

$$\begin{pmatrix} 5 & 4 & 3 \\ 7 & 1 & 3 \\ 9 & 4 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 6 & 6 \\ 4 & 6 & 5 \\ 6 & 7 & 7 \end{pmatrix} = \begin{pmatrix} 44 & 75 & 71 \\ 36 & 69 & 68 \\ 40 & 85 & 81 \end{pmatrix}$$

Terminal Transposta

```
3.000000 4.000000 6.000000
6.000000 7.000000 4.000000
4.000000 4.000000 4.000000

6.000000 4.000000 7.000000
1.000000 6.000000 9.000000
6.000000 1.000000 4.000000

6.000000 1.000000 6.000000
4.000000 6.000000 1.000000
7.000000 9.000000 4.000000

58.000000 42.000000 81.000000
67.000000 70.000000 121.000000
52.000000 44.000000 80.000000
```

Matrix Calculator

$$\begin{pmatrix} 3 & 4 & 6 \\ 6 & 7 & 4 \\ 4 & 4 & 4 \end{pmatrix} \cdot \begin{pmatrix} 6 & 4 & 7 \\ 1 & 6 & 9 \\ 6 & 1 & 4 \end{pmatrix} = \begin{pmatrix} 58 & 42 & 81 \\ 67 & 70 & 121 \\ 52 & 44 & 80 \end{pmatrix}$$

4x4:

Terminal Original`

```
5.000000 2.000000 0.000000 2.000000
1.000000 1.000000 7.000000 7.000000
5.000000 1.000000 5.000000 2.000000
7.000000 7.000000 4.000000 2.000000

1.000000 3.000000 5.000000 4.000000
5.000000 5.000000 1.000000 8.000000
6.000000 3.000000 3.000000 2.000000
0.000000 5.000000 3.000000 8.000000

15.000000 35.000000 33.000000 52.000000
48.000000 64.000000 48.000000 82.000000
40.000000 45.000000 47.000000 54.000000
66.000000 78.000000 60.000000 108.000000
```

Matrix Calculator

$$\begin{pmatrix} 5 & 2 & 0 & 2 \\ 1 & 1 & 7 & 7 \\ 5 & 1 & 5 & 2 \\ 7 & 7 & 4 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 & 3 & 5 & 4 \\ 5 & 5 & 1 & 8 \\ 6 & 3 & 3 & 2 \\ 0 & 5 & 3 & 8 \end{pmatrix} = \begin{pmatrix} 15 & 35 & 33 & 52 \\ 48 & 64 & 48 & 82 \\ 40 & 45 & 47 & 54 \\ 66 & 78 & 60 & 108 \end{pmatrix}$$

Terminal Transposta

```
9.000000 4.000000 8.000000 0.000000
3.000000 4.000000 0.000000 1.000000
8.000000 8.000000 7.000000 4.000000
6.000000 4.000000 7.000000 3.000000

7.000000 9.000000 0.000000 2.000000
5.000000 4.000000 1.000000 9.000000
0.000000 8.000000 0.000000 8.000000
9.000000 1.000000 6.000000 0.000000

7.000000 5.000000 0.000000 9.000000
9.000000 4.000000 8.000000 1.000000
0.000000 1.000000 0.000000 6.000000
2.000000 9.000000 8.000000 0.000000

83.000000 161.000000 4.000000 118.000000
50.000000 44.000000 10.000000 42.000000
132.000000 164.000000 32.000000 144.000000
89.000000 129.000000 22.000000 104.000000
```

Matrix Calculator

$$\begin{pmatrix} 9 & 4 & 8 & 0 \\ 3 & 4 & 0 & 1 \\ 8 & 8 & 7 & 4 \\ 6 & 4 & 7 & 3 \end{pmatrix} \cdot \begin{pmatrix} 7 & 9 & 0 & 2 \\ 5 & 4 & 1 & 9 \\ 0 & 8 & 0 & 8 \\ 9 & 1 & 6 & 0 \end{pmatrix} = \begin{pmatrix} 83 & 161 & 4 & 118 \\ 50 & 44 & 10 & 42 \\ 132 & 164 & 32 & 144 \\ 89 & 129 & 22 & 104 \end{pmatrix}$$

Etapa 03: Mensurar Testes e Comparar Resultados

Planilha Excel Testes Mensurados.xlsx contendo as tabelas com as médias de tempo de execução de cada método para variações de matrizes 200 <= i <= 2000 com passo i+=200 (10 variações para cada tamanho), além do speedup e gráficos comparativos entre os tempos de cada método.

Arquivo teste.bash utilizado para executar repetidas vezes o código cache.c para coletar os tempos estimados.

Speedup	
Desempenho após melhoria	125,9598417
Desempenho antes da melhoria	412,8254468
Resultado	**69%**

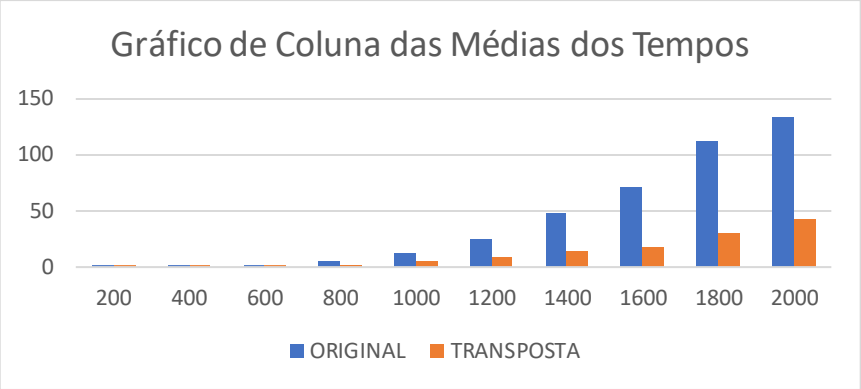
Multiplicação: M1M2 Original e (Transposta M2 + M1M2T)

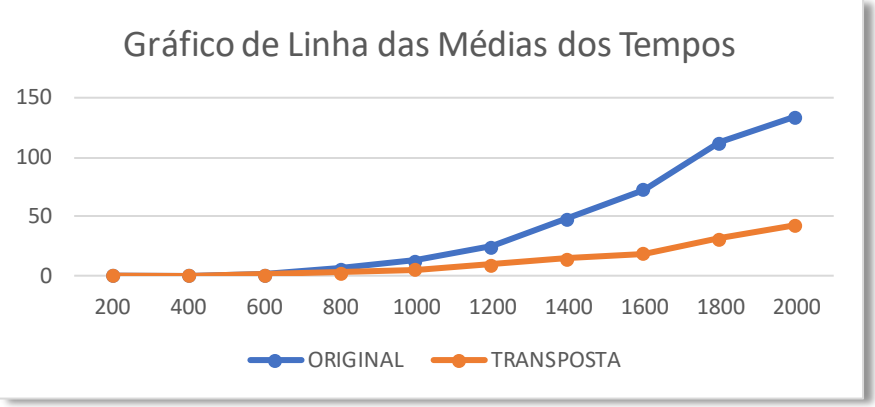
Médias	ORIGINAL	TRANSPOSTA	
200	**0,0504209**	**0,0396566**	**21%**
400	**0,3541346**	**0,3403097**	**4%**
600	**1,2969437**	**1,1589056**	**11%**
800	**6,1276362**	**2,7671768**	**55%**
1000	**12,9871321**	**5,3345566**	**59%**
1200	**24,710141**	**9,4182505**	**62%**
1400	**48,3662243**	**14,8396299**	**69%**
1600	**72,2445991**	**18,426019**	**74%**
1800	**112,4384361**	**31,0008953**	**72%**
2000	**134,2497788**	**42,6344417**	**68%**

Multiplicação: M1M2 Original e M1M2T

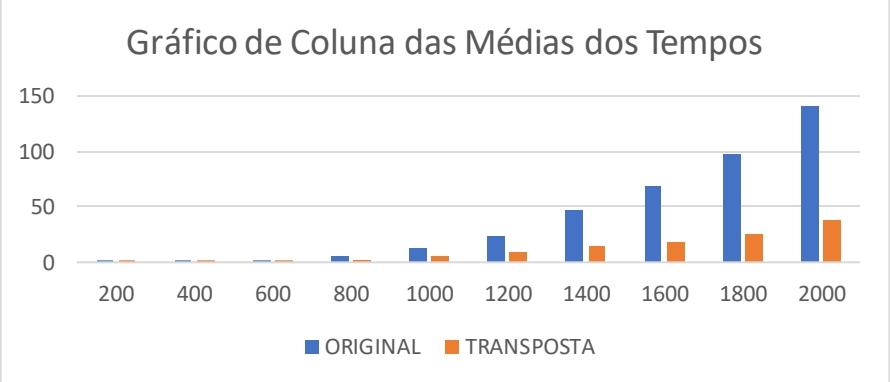
Médias	ORIGINAL	TRANSPOSTA	
200	0,040221	0,0400856	**0%**
400	0,3384745	0,3376051	**0%**
600	1,3130073	1,1273715	**14%**
800	6,4425668	2,4919304	**61%**
1000	13,1249193	5,2870795	**60%**
1200	24,4136586	8,7899106	**64%**
1400	46,3822789	14,351493	**69%**
1600	68,6064353	17,8445037	**74%**
1800	97,5052611	25,9312042	**73%**
2000	141,2555054	37,3131328	**74%**

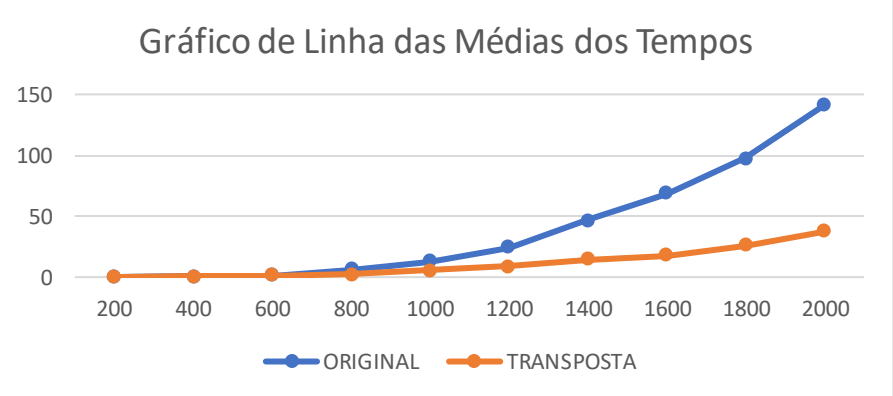
M1M2 Original e (Transposta M2 + M1M2T)





M1M2 Original e M1M2T





Etapa 04: Valores CACHE HIT e CACHE MISS utilizando Valgrind para matrizes 200 <= i < 1500 com i+= 200.

L1 misses: miss de instrução em L1

LLi misses: miss de instrução em L2

D1 misses: miss de dados em L1

LLi misses: miss de dados em L2

LL refs: número total de referências ao último nível de cache (leituras e escritas);

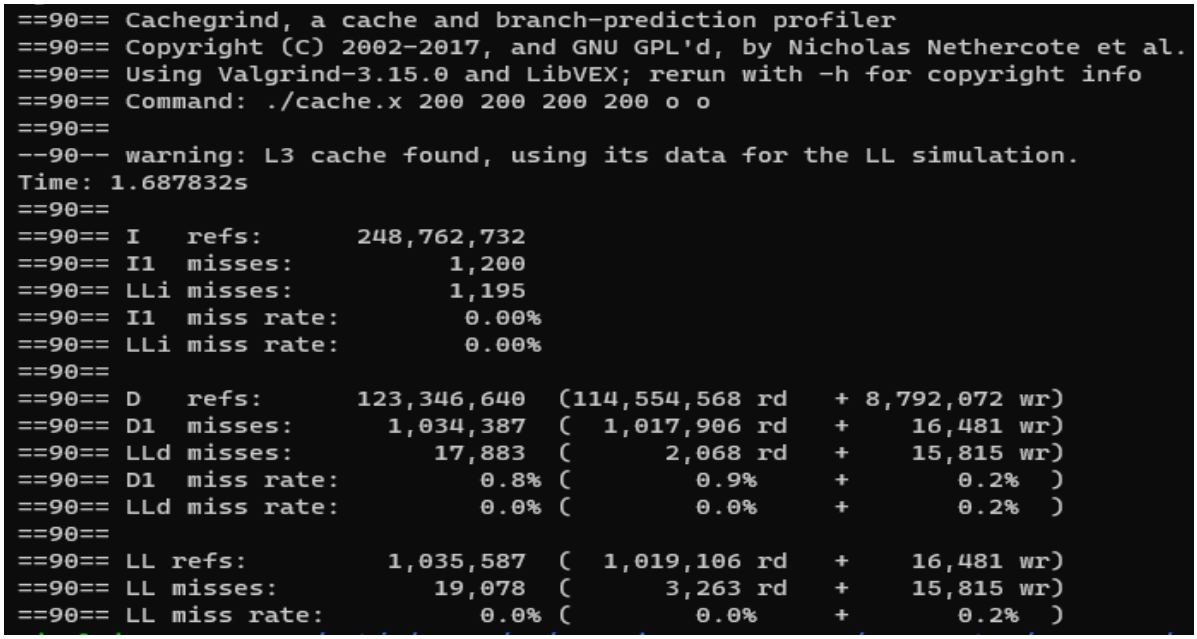
LL misses: número total de misses de dados e instruções ao último nível de cache (leitura e escrita);

LL miss rate: taxa de cache miss ao último nível de cache

Cache miss para leitura ou escrita = (LL misses / LL refs)

Cache hit para leitura ou escrita = (LL refs – LL misses) / LL refs

Matriz Original 200x200:



```
==90== Cachegrind, a cache and branch-prediction profiler
==90== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==90== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==90== Command: ./cache.x 200 200 200 200 o o
==90==
--90-- warning: L3 cache found, using its data for the LL simulation.
Time: 1.687832s
==90==
==90== I   refs:      248,762,732
==90== I1  misses:          1,200
==90== LLi misses:          1,195
==90== I1  miss rate:        0.00%
==90== LLi miss rate:        0.00%
==90==
==90== D   refs:      123,346,640  (114,554,568 rd   + 8,792,072 wr)
==90== D1  misses:      1,034,387  (  1,017,906 rd   +    16,481 wr)
==90== LLd misses:         17,883  (      2,068 rd   +    15,815 wr)
==90== D1  miss rate:         0.8% (        0.9%     +       0.2%  )
==90== LLd miss rate:         0.0% (        0.0%     +       0.2%  )
==90==
==90== LL refs:         1,035,587  (  1,019,106 rd   +    16,481 wr)
==90== LL misses:          19,078  (      3,263 rd   +    15,815 wr)
==90== LL miss rate:          0.0% (        0.0%     +       0.2%  )
```

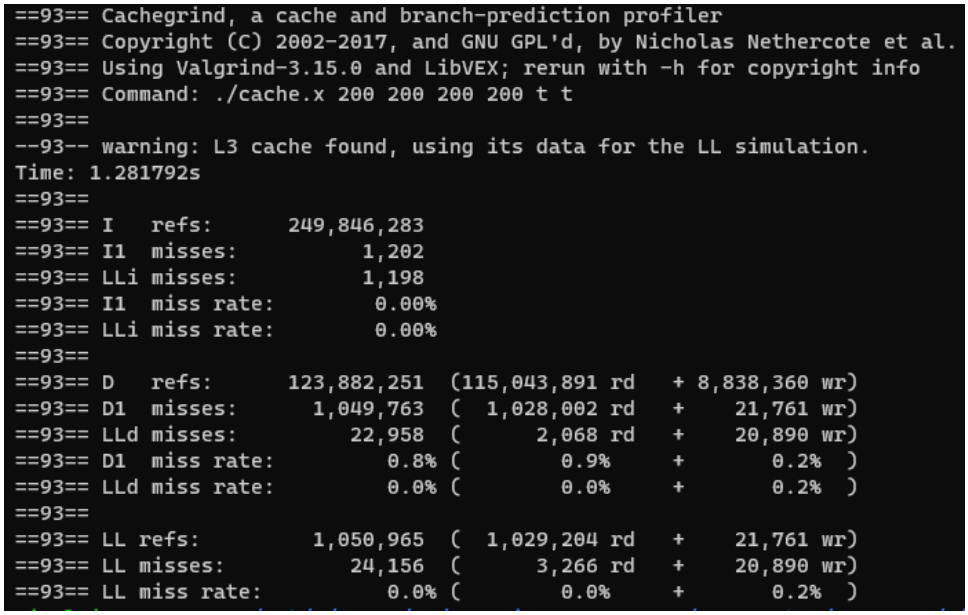
CACHE MISS = (LL misses / LL refs) * 100

(19.078 / 1.035.597) * 100 = 1,842240198071238823971332200964 3

CACHE HIT = ((LL refs – LL misses) / LL refs) * 100

((1.035.587 – 19.078) / 1.035.587) * 100 = 98,157759801928761176028667799036

Matriz Transposta 200x200:



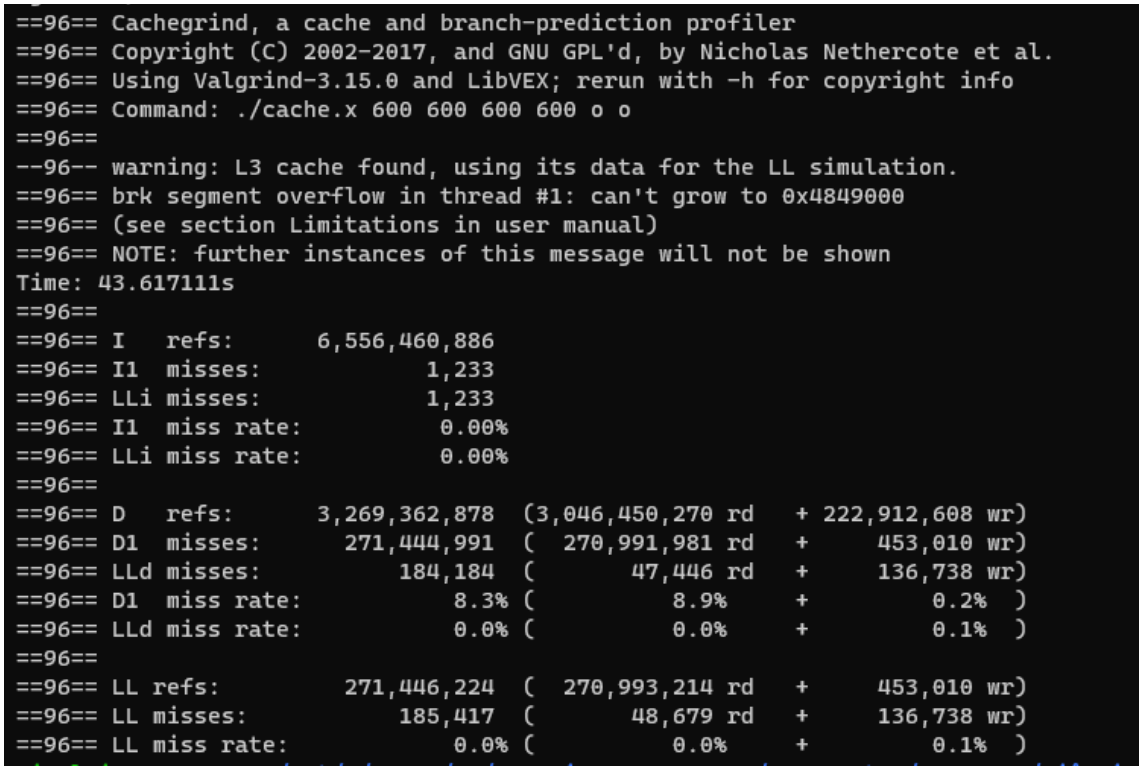
CACHE MISS = (LL misses / LL refs) * 100

(24.156 / 1.050.965) * 100 = 2,298459035267587407763341310 1293

CACHE HIT = ((LL refs – LL misses) / LL refs) * 100

((1.050.965 – 24.156) / 1.050.965) * 100 = 97,701540964732412592236658689871

Matriz Original 600x600:



```
==96== Cachegrind, a cache and branch-prediction profiler
==96== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==96== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==96== Command: ./cache.x 600 600 600 600 o o
==96==
--96-- warning: L3 cache found, using its data for the LL simulation.
==96== brk segment overflow in thread #1: can't grow to 0x4849000
==96== (see section Limitations in user manual)
==96== NOTE: further instances of this message will not be shown
Time: 43.617111s
==96==
==96== I   refs:      6,556,460,886
==96== I1  misses:            1,233
==96== LLi misses:            1,233
==96== I1  miss rate:          0.00%
==96== LLi miss rate:          0.00%
==96==
==96== D   refs:      3,269,362,878  (3,046,450,270 rd   + 222,912,608 wr)
==96== D1  misses:      271,444,991  (  270,991,981 rd   +     453,010 wr)
==96== LLd misses:          184,184  (       47,446 rd   +     136,738 wr)
==96== D1  miss rate:           8.3% (          8.9%     +         0.2%  )
==96== LLd miss rate:           0.0% (          0.0%     +         0.1%  )
==96==
==96== LL refs:         271,446,224  (  270,993,214 rd   +     453,010 wr)
==96== LL misses:           185,417  (       48,679 rd   +     136,738 wr)
==96== LL miss rate:            0.0% (          0.0%     +         0.1%  )
```

CACHE MISS = (LL misses / LL refs) * 100

(185.417 / 271.446.224) * 100 = 0,06830708391066069867304545743101

CACHE HIT = ((LL refs – LL misses) / LL refs) * 100

((271.446.224 – 185.417) / 271.446.224) * 100 = 99,93169291608933930 1326954542569

Matriz Transposta 600x600:

```
==97== Cachegrind, a cache and branch-prediction profiler
==97== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==97== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==97== Command: ./cache.x 600 600 600 600 t t
==97==
--97-- warning: L3 cache found, using its data for the LL simulation.
==97== brk segment overflow in thread #1: can't grow to 0x4849000
==97== (see section Limitations in user manual)
==97== NOTE: further instances of this message will not be shown
Time: 35.214424s
==97==
==97== I   refs:      6,565,949,056
==97== I1  misses:            1,263
==97== LLi misses:            1,263
==97== I1  miss rate:          0.00%
==97== LLi miss rate:          0.00%
==97==
==97== D   refs:      3,274,089,662  (3,050,797,615 rd   + 223,292,047 wr)
==97== D1  misses:       27,773,404  (   27,589,401 rd   +     184,003 wr)
==97== LLd misses:          270,664  (       88,426 rd   +     182,238 wr)
==97== D1  miss rate:           0.8% (          0.9%     +         0.1%  )
==97== LLd miss rate:           0.0% (          0.0%     +         0.1%  )
==97==
==97== LL refs:          27,774,667  (   27,590,664 rd   +     184,003 wr)
==97== LL misses:           271,927  (       89,689 rd   +     182,238 wr)
==97== LL miss rate:            0.0% (          0.0%     +         0.1%  )
```

CACHE MISS = (LL misses / LL refs) * 100

(271.927 / 27.774.667) * 100 = 0,979046841497685642819768100 19

CACHE HIT = ((LL refs – LL misses) / LL refs) * 100

((27.774.667 – 271.927) / 27.774.667) * 100 = 99,020953158502314357180231899 81

Matriz Original 1400x1400:

```
==104== Cachegrind, a cache and branch-prediction profiler
==104== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==104== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==104== Command: ./cache.x 1400 1400 1400 1400 o o
==104==
--104-- warning: L3 cache found, using its data for the LL simulation.
==104== brk segment overflow in thread #1: can't grow to 0x4846000
==104== (see section Limitations in user manual)
==104== NOTE: further instances of this message will not be shown
Time: 667.576050s
==104==
==104== I   refs:      82,734,187,690
==104== I1  misses:             1,264
==104== LLi misses:             1,264
==104== I1  miss rate:           0.00%
==104== LLi miss rate:           0.00%
==104==
==104== D   refs:      41,319,157,928  (38,537,761,947 rd   + 2,781,395,981 wr)
==104== D1  misses:     3,409,390,241  ( 3,406,934,037 rd   +     2,456,204 wr)
==104== LLd misses:       358,609,146  (   357,702,380 rd   +       906,766 wr)
==104== D1  miss rate:            8.3% (           8.8%     +           0.1%  )
==104== LLd miss rate:            0.9% (           0.9%     +           0.0%  )
==104==
==104== LL refs:        3,409,391,505  ( 3,406,935,301 rd   +     2,456,204 wr)
==104== LL misses:        358,610,410  (   357,703,644 rd   +       906,766 wr)
==104== LL miss rate:             0.3% (           0.3%     +           0.0%  )
```

CACHE MISS = (LL misses / LL refs) * 100

(358.610.410 / 3.409.391.505) * 100 = 10,518311243343113802942381649 42

CACHE HIT = ((LL refs – LL misses) / LL refs) * 100

((3.409.391.505 – 358.610.410) / 3.409.391.505) * 100 = 89,481688756656886197057618350 58

Matriz Transposta 1400x1400:

```
==105== Cachegrind, a cache and branch-prediction profiler
==105== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==105== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==105== Command: ./cache.x 1400 1400 1400 1400 t t
==105==
--105-- warning: L3 cache found, using its data for the LL simulation.
==105== brk segment overflow in thread #1: can't grow to 0x4846000
==105== (see section Limitations in user manual)
==105== NOTE: further instances of this message will not be shown
Time: 426.212280s
==105==
==105== I   refs:      82,785,455,328
==105== I1  misses:             1,267
==105== LLi misses:             1,267
==105== I1  miss rate:           0.00%
==105== LLi miss rate:           0.00%
==105==
==105== D   refs:      41,344,749,579  (38,561,346,429 rd   + 2,783,403,150 wr)
==105== D1  misses:       347,197,272  (   346,206,919 rd   +       990,353 wr)
==105== LLd misses:       345,243,373  (   344,255,319 rd   +       988,054 wr)
==105== D1  miss rate:            0.8% (           0.9%     +           0.0%  )
==105== LLd miss rate:            0.8% (           0.9%     +           0.0%  )
==105==
==105== LL refs:          347,198,539  (   346,208,186 rd   +       990,353 wr)
==105== LL misses:        345,244,640  (   344,256,586 rd   +       988,054 wr)
==105== LL miss rate:             0.3% (           0.3%     +           0.0%  )
```

CACHE MISS = (LL misses / LL refs) * 100

(345.244.640 / 347.198.539) * 100 = 99,437238703357562227529995453 12

CACHE HIT = ((LL refs – LL misses) / LL refs) * 100

((347.198.539 – 345.244.640) / 347.198.539) * 100 = 0,562761296642437772470004546879 73