USANDO VALGRIND Y CACHEGRIND PROBANDO

con Valgrind con la opción de Cachegrind se obtuvo lo siguiente:

• Con la multiplicación de matrices de 3 bucles, como se muestra

```
==10940== Cachegrind, a cache and branch-prediction profiler
==10940== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==10940== Using Valgrind-3.14.0.SVN and LibVEX; rerun with -h for copyright info
==10940== Command: ./3b
==10940==
--10940-- warning: L3 cache found, using its data for the LL simulation.
--10940-- run: /usr/bin/dsymutil "./3b"
warning: no debug symbols in executable (-arch x86_64)
Matriz de 200x200

tiempo: 2.03401
==10940==
==10940== I   refs:      234,795,397
==10940== I1  misses:          5,198
==10940== LLi misses:          2,924
==10940== I1  miss rate:        0.00%
==10940== LLi miss rate:        0.00%
==10940==
==10940== D   refs:      196,220,878  (178,906,985 rd   + 17,313,893 wr)
==10940== D1  misses:        538,088  (    527,949 rd   +     10,139 wr)
==10940== LLd misses:         17,462  (      8,206 rd   +      9,256 wr)
==10940== D1  miss rate:         0.3% (        0.3%     +        0.1%  )
==10940== LLd miss rate:         0.0% (        0.0%     +        0.1%  )
==10940==
==10940== LL refs:           543,286  (    533,147 rd   +     10,139 wr)
==10940== LL misses:          20,386  (     11,130 rd   +      9,256 wr)
==10940== LL miss rate:          0.0% (        0.0%     +        0.1%  )
```

• Con la multiplicación de matrices de 6 bucles, como se muestra

```
==11023== Cachegrind, a cache and branch-prediction profiler
==11023== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==11023== Using Valgrind-3.14.0.SVN and LibVEX; rerun with -h for copyright info
==11023== Command: ./6b
==11023==
--11023-- warning: L3 cache found, using its data for the LL simulation.
--11023-- run: /usr/bin/dsymutil "./6b"
warning: no debug symbols in executable (-arch x86_64)
Matriz de 200x200

tiempo: 4.13469
==11023==
==11023== I   refs:      463,897,554
==11023== I1  misses:          5,309
==11023== LLi misses:          2,954
==11023== I1  miss rate:        0.00%
==11023== LLi miss rate:        0.00%
==11023==
==11023== D   refs:      382,417,686  (280,707,165 rd   + 101,710,521 wr)
==11023== D1  misses:         78,480  (     70,854 rd   +       7,626 wr)
==11023== LLd misses:         17,472  (     10,722 rd   +       6,750 wr)
==11023== D1  miss rate:         0.0% (        0.0%     +         0.0%  )
==11023== LLd miss rate:         0.0% (        0.0%     +         0.0%  )
==11023==
==11023== LL refs:            83,789  (     76,163 rd   +       7,626 wr)
==11023== LL misses:          20,426  (     13,676 rd   +       6,750 wr)
==11023== LL miss rate:          0.0% (        0.0%     +         0.0%  )
```

El BlockSize representa el tamaño del bloque que debe calcularse para cada iteración. Esto no debería ser más grande que el tamaño de la matriz. Así también se hizo la prueba con diferentes tamaño de bloque:

• BlockSize = 80 con una matriz de 200x200.

```
Matriz de 200x200

tiempo: 3.94182
==997==
==997== I   refs:      459,984,522
==997== I1  misses:          5,253
==997== LLi misses:          2,958
==997== I1  miss rate:        0.00%
==997== LLi miss rate:        0.00%
==997==
==997== D   refs:      378,088,341  (278,209,461 rd   + 99,878,880 wr)
==997== D1  misses:        167,319  (    159,692 rd   +      7,627 wr)
==997== LLd misses:         17,465  (     10,721 rd   +      6,744 wr)
==997== D1  miss rate:         0.0% (        0.1%     +        0.0%  )
==997== LLd miss rate:         0.0% (        0.0%     +        0.0%  )
==997==
==997== LL refs:           172,572  (    164,945 rd   +      7,627 wr)
==997== LL misses:          20,423  (     13,679 rd   +      6,744 wr)
==997== LL miss rate:          0.0% (        0.0%     +        0.0%  )
```

• BlockSize = 60 con una matriz de 800x800

```
Matriz de 800x800

tiempo: 283.827
==699==
==699== I   refs:      28,839,764,745
==699== I1  misses:             5,313
==699== LLi misses:             3,199
==699== I1  miss rate:           0.00%
==699== LLi miss rate:           0.00%
==699==
==699== D   refs:      24,049,342,284  (17,697,451,937 rd   + 6,351,890,347 wr)
==699== D1  misses:       123,987,704  (   123,152,720 rd   +       834,984 wr)
==699== LLd misses:           171,495  (        88,926 rd   +        82,569 wr)
==699== D1  miss rate:            0.5% (           0.7%     +           0.0%  )
==699== LLd miss rate:            0.0% (           0.0%     +           0.0%  )
==699==
==699== LL refs:          123,993,017  (   123,158,033 rd   +       834,984 wr)
==699== LL misses:            174,694  (        92,125 rd   +        82,569 wr)
==699== LL miss rate:             0.0% (           0.0%     +           0.0%  )
```

• BlockSize = 64 con una matriz de 800x800.

```
Matriz de 800x800

tiempo: 289.18
==834==
==834== I   refs:      28,803,205,719
==834== I1  misses:             5,259
==834== LLi misses:             3,211
==834== I1  miss rate:           0.00%
==834== LLi miss rate:           0.00%
==834==
==834== D   refs:      24,014,922,776  (17,677,594,183 rd   + 6,337,328,593 wr)
==834== D1  misses:       192,464,510  (   191,305,909 rd   +     1,158,601 wr)
==834== LLd misses:           171,556  (        88,940 rd   +        82,616 wr)
==834== D1  miss rate:            0.8% (           1.1%     +           0.0%  )
==834== LLd miss rate:            0.0% (           0.0%     +           0.0%  )
==834==
==834== LL refs:          192,469,769  (   191,311,168 rd   +     1,158,601 wr)
==834== LL misses:            174,767  (        92,151 rd   +        82,616 wr)
==834== LL miss rate:             0.0% (           0.0%     +           0.0%  )
```

• BlockSize = 72 con una matriz de 800x800.

```
Matriz de 800x800

tiempo: 320.111
==878==
==878== I   refs:      28,743,343,667
==878== I1  misses:             5,316
==878== LLi misses:             3,212
==878== I1  miss rate:           0.00%
==878== LLi miss rate:           0.00%
==878==
==878== D   refs:      23,980,588,455  (17,657,785,283 rd   + 6,322,803,172 wr)
==878== D1  misses:       515,656,867  (   514,587,911 rd   +     1,068,956 wr)
==878== LLd misses:           171,568  (        88,987 rd   +        82,581 wr)
==878== D1  miss rate:            2.2% (           2.9%     +           0.0%  )
==878== LLd miss rate:            0.0% (           0.0%     +           0.0%  )
==878==
==878== LL refs:          515,662,183  (   514,593,227 rd   +     1,068,956 wr)
==878== LL misses:            174,780  (        92,199 rd   +        82,581 wr)
==878== LL miss rate:             0.0% (           0.0%     +           0.0%  )
```

• BlockSize = 84 con una matriz de 800x800.

```
Matriz de 800x800

tiempo: 323.515
==900==
==900== I   refs:      28,678,514,794
==900== I1  misses:             5,255
==900== LLi misses:             3,226
==900== I1  miss rate:           0.00%
==900== LLi miss rate:           0.00%
==900==
==900== D   refs:      23,912,176,128  (17,618,316,349 rd   + 6,293,859,779 wr)
==900== D1  misses:       497,616,611  (   496,726,478 rd   +       890,133 wr)
==900== LLd misses:           174,150  (        91,526 rd   +        82,624 wr)
==900== D1  miss rate:            2.1% (           2.8%     +           0.0%  )
==900== LLd miss rate:            0.0% (           0.0%     +           0.0%  )
==900==
==900== LL refs:          497,621,866  (   496,731,733 rd   +       890,133 wr)
==900== LL misses:            177,376  (        94,752 rd   +        82,624 wr)
==900== LL miss rate:             0.0% (           0.0%     +           0.0%  )
```

• BlockSize = 128 con una matriz de 800x800.

```
Matriz de 800x800

tiempo: 249.819
==973==
==973== I   refs:      28,539,256,448
==973== I1  misses:             5,260
==973== LLi misses:             3,173
==973== I1  miss rate:           0.00%
==973== LLi miss rate:           0.00%
==973==
==973== D   refs:      23,810,194,476  (17,559,480,256 rd   + 6,250,714,220 wr)
==973== D1  misses:       504,301,749  (   503,680,755 rd   +       620,994 wr)
==973== LLd misses:           198,416  (       115,831 rd   +        82,585 wr)
==973== D1  miss rate:            2.1% (           2.9%     +           0.0%  )
==973== LLd miss rate:            0.0% (           0.0%     +           0.0%  )
==973==
==973== LL refs:          504,307,009  (   503,686,015 rd   +       620,994 wr)
==973== LL misses:            201,589  (       119,004 rd   +        82,585 wr)
==973== LL miss rate:             0.0% (           0.0%     +           0.0%  )
```

CONCLUSIONES

En la memoria cache al utilizarla con una gran cantidad de datos tiene un efecto de mejora muy significativa, optimizando el tiempo de ejecución del programa. Así, se puede ver que la diferencia de cache misses de la multiplicación de 6 bucles que es

en bloques con respecto a la multiplicación de 3 bucles corresponde a casi mas de la mitad.