Memoria Practica 6: Caché

David Martínez Díaz GII-ADE

Modificación de los ficheros .cc:

En primer lugar, en el fichero line.cc he añadido a la línea de código que hay completar la siguiente expresión:

- ➔ Bytes[i]++; (Una operación que se realiza en el vector de Bytes, donde incrementa la posición de i, de 1 en 1). Para saber el tamaño de línea de nuestra caché debemos realizar un incremento en el que se realiza una función XOR, una operación sencilla para no ocupar mucho tiempo de ejecución.

En segundo lugar, en el fichero size.cc, le he añadido la siguiente línea de código:

- ➔ Bytes[(64*i)&(size-1)]++; (Una operación que incrementa la posición del vector de bytes). Con los niveles de caché ocurre algo parecido, se va almacenando valores en un vector modificando cada línea de cache de forma que cuando alcanzamos el ultimo valor volvemos al inicio.

Como podemos ver en estas capturas para line.cc:

```cpp
#include <algorithm>    // nth_element
#include <array>        // array
#include <chrono>       // high_resolution_clock
#include <iomanip>      // setw
#include <iostream>     // cout
#include <vector>       // vector

using namespace std::chrono;

const unsigned MAXLINE = 1024; // maximun line size to test
const unsigned GAP = 12;       // gap for cout columns
const unsigned REP = 100;      // number of repetitions of every test

int main()
{
        std::cout << "#"
                  << std::setw(GAP - 1) << "line (B)"
                  << std::setw(GAP    ) << "time (µs)"
                  << std::endl;

        for (unsigned line = 1; line <= MAXLINE; line <<= 1) // line in bytes
        {
                std::vector<duration<double, std::micro>> score(REP);

                for (auto &s: score)
                {
                        std::vector<char> bytes(1 << 24); // 16MB

                        auto start = high_resolution_clock::now();

                        for (unsigned i = 0; i < bytes.size(); i += line)
                                bytes[i]++;

                        auto stop = high_resolution_clock::now();

                        s = stop - start;
                }

                std::nth_element(score.begin(),
                                 score.begin() + score.size() / 2,
                                 score.end());

                std::cout << std::setw(GAP) << line
                          << std::setw(GAP) << std::fixed << std::setprecision(1)
                          << std::setw(GAP) << score[score.size() / 2].count()
                          << std::endl;
        }
```

Y para size.cc:

```cpp
#include <algorithm>    // nth_element
#include <array>        // array
#include <chrono>       // high_resolution_clock
#include <iomanip>      // setw
#include <iostream>     // cout
#include <vector>       // vector

using namespace std::chrono;

const unsigned MINSIZE = 1 << 10; // minimun line size to test:  1KB
const unsigned MAXSIZE = 1 << 26; // maximun line size to test: 32MB
const unsigned GAP = 12;          // gap for cout columns
const unsigned REP = 100;         // number of repetitions of every test
const unsigned STEPS = 1e6;       // steps

int main()
{
        std::cout << "#"
                  << std::setw(GAP - 1) << "line (B)"
                  << std::setw(GAP    ) << "time (µs)"
                  << std::endl;

        for (unsigned size = MINSIZE; size <= MAXSIZE; size *= 2)
        {
                std::vector<duration<double, std::micro>> score(REP);

                for (auto &s: score)
                {
                        std::vector<char> bytes(size);

                        auto start = high_resolution_clock::now();SS

                        for (unsigned i = 0; i < STEPS; ++i)
                                bytes[(i*64)&(size-1)]++;

                        auto stop = high_resolution_clock::now();

                        s = stop - start;
                }

                std::nth_element(score.begin(),
                                 score.begin() + score.size() / 2,
                                 score.end());

                std::cout << std::setw(GAP) << size
                          << std::setw(GAP) << std::fixed << std::setprecision(1)
                          << std::setw(GAP) << score[score.size() / 2].count()
```

Una vez hemos modificado el código simplemente tenemos que utilizar el comando “make” en la terminal y se generara todos los ejecutables:

```
dmartinez01@dmartinez01-VirtualBox:~/Escritorio/Practica 6$ make
g++ -march=native -Ofast -std=c++11 -Wall    line.cc   -o line
g++ -march=native -Ofast -std=c++11 -Wall    size.cc   -o size
./line | tee line.dat
#   line (B)  time (µs)
          1      8985.9
          2      4394.3
          4      2321.4
          8      1514.1
         16      1212.3
         32      1017.2
         64       990.3
        128       783.8
        256       563.7
        512       282.9
       1024       136.5
./size | tee size.dat
#   line (B)  time (µs)
       1024       512.3
       2048       510.7
       4096       510.3
       8192       513.3
      16384       512.2
      32768       512.7
      65536       888.4
     131072       888.7
     262144      1059.9
     524288      1471.8
    1048576      1477.9
    2097152      1515.5
    4194304      1501.5
    8388608      2049.5
   16777216      4028.1
   33554432      4914.5
   67108864      4949.4
```

Por otro lado, para ver que procesador tenemos y diferentes características de nuestro ordenador utilizaremos la orden “lscpu”:

```
dmartinez01@dmartinez01-VirtualBox:~/Escritorio/Practica 6$ lscpu
Arquitectura:                        x86_64
modo(s) de operación de las CPUs:    32-bit, 64-bit
Orden de los bytes:                  Little Endian
CPU(s):                              4
Lista de la(s) CPU(s) en línea:      0-3
Hilo(s) de procesamiento por núcleo: 1
Núcleo(s) por «socket»:              4
«Socket(s)»                          1
Modo(s) NUMA:                        1
ID de fabricante:                    GenuineIntel
Familia de CPU:                      6
Modelo:                              158
Nombre del modelo:                   Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
Revisión:                            10
CPU MHz:                             2592.000
BogoMIPS:                            5184.00
Fabricante del hipervisor:           KVM
Tipo de virtualización:              lleno
Caché L1d:                           32K
Caché L1i:                           32K
Caché L2:                            256K
Caché L3:                            12288K
CPU(s) del nodo NUMA 0:              0-3
```

Por ultimo mostraremos los datos para cada ejecutable:

➔ *Line.dat:*

 # line (B) time (µs)

1 8985.9

2 4394.3

4 2321.4

8 1514.1

16 1212.3

32 1017.2

64 990.3

128 783.8

256 563.7

512 282.9

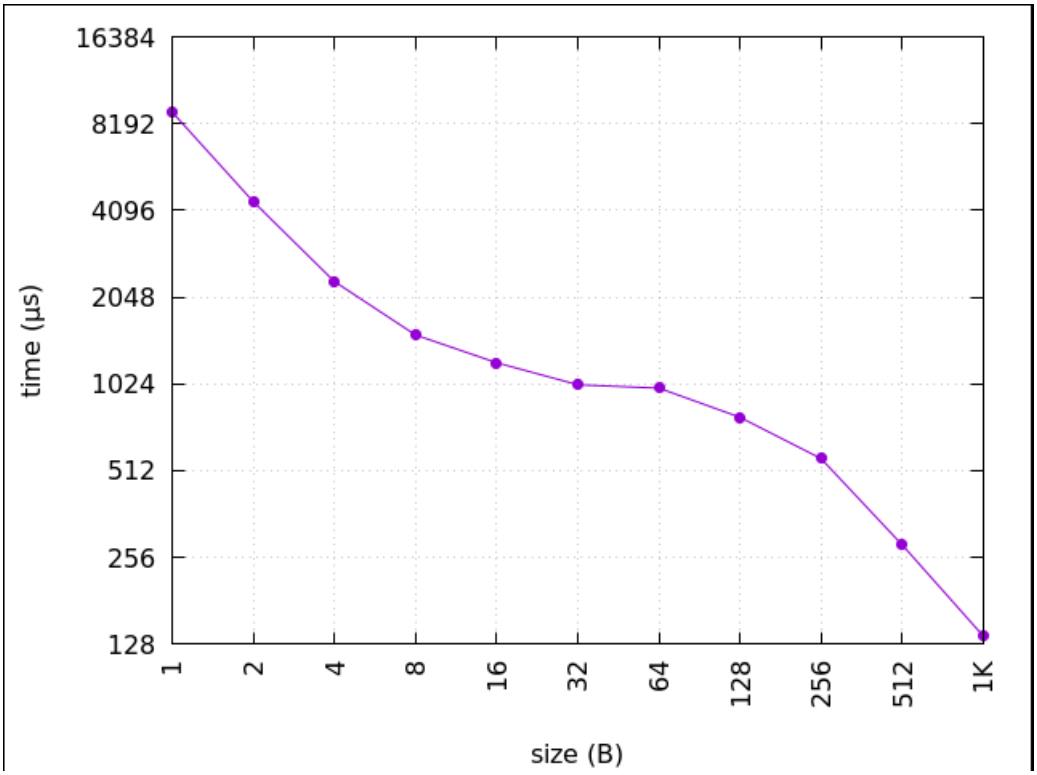
1024 136.5

➔ *Size.dat:*

# line (B)	time (µs)
1024	512.3
2048	510.7
4096	510.3
8192	513.3
16384	512.2
32768	512.7
65536	888.4
131072	888.7
262144	1059.9
524288	1471.8
1048576	1477.9
2097152	1515.5
4194304	1501.5
8388608	2049.5
16777216	4028.1
33554432	4914.5
67108864	4949.4

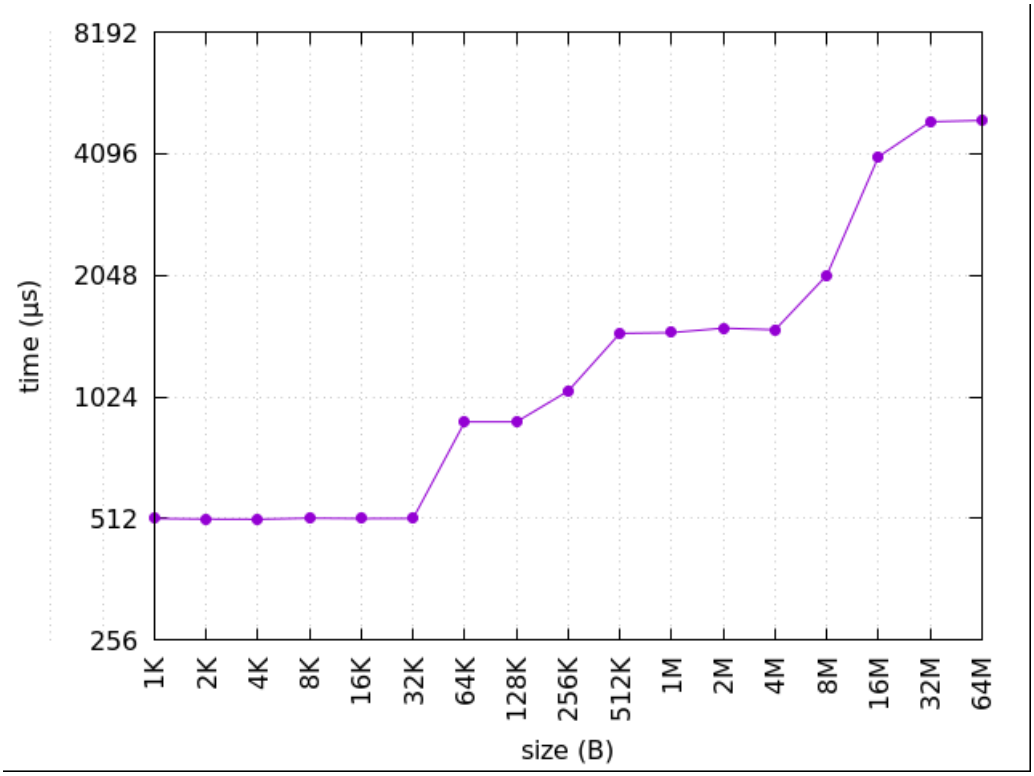
Y también mostraremos sus respectivas graficas:

➔ *Line:*



Y como podemos observar en esta gráfica, si cuando el tamaño es 64B, la línea realiza una bajada progresiva, lo que verifica que el tamaño de línea es de 64B. Además, en esta gráfica en cada iteración del bucle se debería de ir reduciendo a la mitad el tiempo de acceso a memoria dando lugar a una gráfica lineal, pero como podemos observar en la gráfica, cuando el tamaño de dato es de 4B, la línea de la gráfica varía.

➔ *Size:*



De esta imagen podemos concluir que en 32KB hay un crecimiento de la línea pronunciada, lo que provoca que la cache es de 32KB.
Por otro lado, entre 64KB y 2MB hay un cierto crecimiento, en el que se aprecia un salto de 128KB a 512KB, lo que supone que la cache L2 es de 256KB. A partir de aquí, se ve un salto muy pronunciado entre 2MB y 4MB, y si sabemos que la cache L3 está en ese intervalo, decimos que su tamaño es de 3MB.

Por último, observamos que L3 es estable, es decir, que necesita el mismo tiempo para traer todos los datos sin importar su tamaño. L3 puede traer datos de hasta 6144KB (6MB), con lo que ahí aparece un salto hacia memoria principal.