DOCUMENTACIÓN PRÁCTICA 5

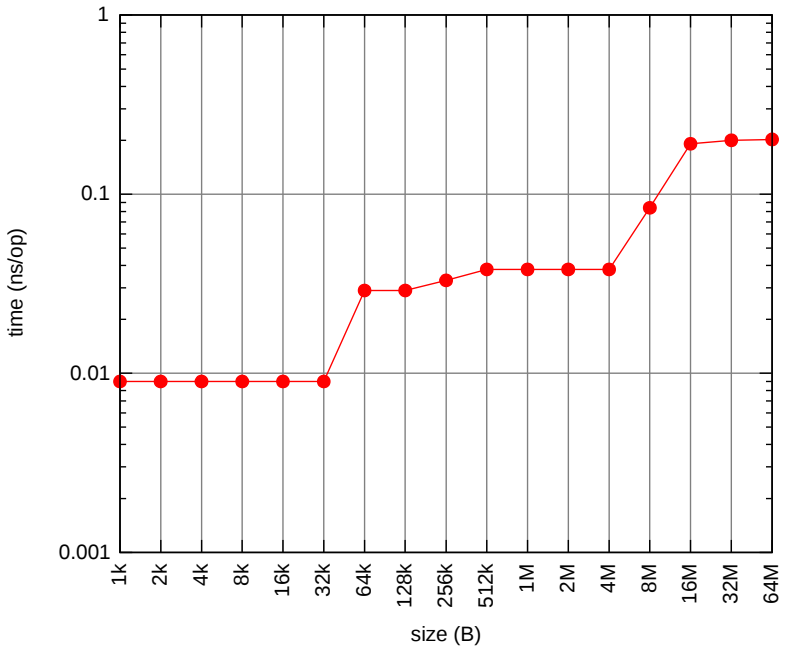
CARACTERÍSTICAS DEL MICROPROCESADOR

Mi modelo de micropocesador es un Intel(R) Core(TM) i7-5700HQ CPU @ 2.70GHz, pero por motivos que tengo una máquina virtual instalada en el pc, he tenido que realizar las pruebas y las gráficas desde el ordenador del aula, el cuál consta con un microprocesador Intel(R) Core(TM) i5-4460 CPU @ 3.20GHz

General information	
Vendor:	GenuineIntel
Processor name (BIOS):	Intel(R) Core(TM) i5-4460 CPU @ 3.20GHz
Cores:	4
Logical processors:	4
Processor type:	Original OEM Processor
CPUID signature:	306C3
Family:	6 (06h)
Model:	60 (03Ch)
Stepping:	3 (03h)
TLB/Cache details:	64-byte Prefetching Data TLB: 1-GB pages, 4-way set associative, 4 entries Data TLB: 4-KB Pages, 4-way set associative, 64 entries Instruction TLB: 4-KByte pages, 8-way set associative, 128 entries L2 TLB: 1-MB, 4-way set associative, 64-byte line size Shared 2nd-Level TLB: 4-KByte / 2-MB pages, 8-way associative, 1024 entries

Cache:	L1 data	L1 instruction	L2	L3
Size:	4 x 32 KB	4 x 32 KB	4 x 256 KB	6 MB
Associativity:	8-way set associative	8-way set associative	8-way set associative	12-way set associative
Line size:	64 bytes	64 bytes	64 bytes	64 bytes
Comments:	Direct-mapped	Direct-mapped	Non-inclusive Direct-mapped	Inclusive Shared between all cores

En la imagen anterior, vemos la capacidad de cada nivel de caché que contiene el procesador.



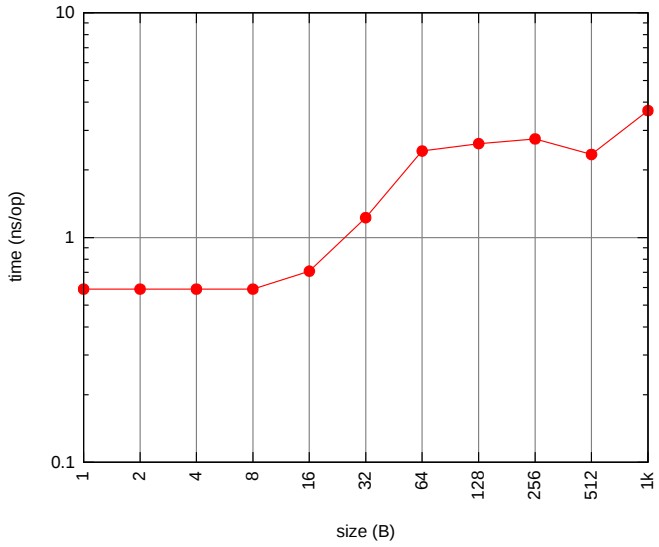
- Esta gráfica corresponde a los resultados obtenidos del programa size.cc.
- El salto se debe a que el tamaño del vector no cabe en el primer nivel de memoria caché del procesador, y comienza a dar fallos por lo que se dispara la velocidad, se carga el vector en el siguiente nivel de memoria caché que se asegura que cabe el tamaño del vector y ocurre lo mismo, comienza a dar fallos y la velocidad

se dispara, así podemos ver los distintos niveles de caché que contiene nuestro microprocesador y el tamaño, hasta que llegamos a la memoria principal.

- Como podemos observar, y coincide con las características del microprocesador, el primer salto lo obtenemos a partir de los 32k, por lo que podemos decir que nuestro primer nivel de memoria caché L1 es de tamaño 32k.
- Al igual que el anterior, ahora lo tenemos el salto entre 128 y 256k aprox. El cuál es nuestro nivel L2 de caché.
- Y por último el tercer nivel que vemos en la gráfica que se encuentra entre 4 y 8M.

```
 3 #include <algorithm>    // nth_element
 4 #include <chrono>       // high_resolution_clock
 5 #include <iomanip>      // setw
 6 #include <iostream>     // cout
 7 #include <vector>       // vector
 8
 9 int main()
10 {
11         const unsigned STEPS = 100000000;
12         const unsigned GAP = 12;
13         const unsigned REP = 50;
14
15         std::cout << "#"
16                   << std::setw(GAP - 1) << "size"
17                   << std::setw(GAP    ) << "time"
18                   << std::endl;
19
20         for (unsigned size = 1 << 10; size <= 1 << 26; size <<= 1)
21         {
22                 using namespace std::chrono;
23
24                 unsigned size_1 = size - 1;
25
26                 std::vector<unsigned char> bits(size);
27
28                 std::vector<double> sec(REP);
29                 for (auto &s: sec)
30                 {
31                         auto start = high_resolution_clock::now();
32                         for (unsigned i=0; i<STEPS; i+=64){
33                                 bits.at(i & size_1)++;
34                         }
35                         auto stop = high_resolution_clock::now();
36                         s = duration_cast<nanoseconds>(stop - start).count() / double(STEPS);
37                 }
38
39                 nth_element(sec.begin(), sec.begin() + sec.size() / 2, sec.end());
40                 std::cout << std::setw(GAP) << size
41                           << std::setw(GAP) << std::fixed << std::setprecision(3)
42                           << sec[sec.size() / 2]
43                           << std::endl;
44         }
45 }
```

Aquí tenemos el código que pertenece al programa line.cc. Sólo le he modificado y añadido un bucle anidado, para que recorra en cada interacción desde la posición 0 hasta el tamaño del vector, y realice una escritura del vector, para así poder tomar mejor los resultados.



- Esta gráfica correspondiente a los datos recogidos de la ejecución del programa line.cc, que mostraré el código en el siguiente apartado.
- Esta gráfica podemos ver claramente dos saltos en la recta conforme el tiempo, es decir, aproximadamente en el tamaño de 16k observamos un incremento del tiempo, y aproximadamente en 512k observamos el segundo incremento en el tiempo.

```
 3 #include <algorithm>    // nth_element
 4 #include <chrono>       // high_resolution_clock
 5 #include <iomanip>      // setw
 6 #include <iostream>     // cout
 7 #include <vector>       // vector
 8
 9 int main()
10 {
11         const unsigned LINE = 1 << 10;  // 1024B
12         const unsigned SIZE = 1 << 21;  // 2MB
13         const unsigned GAP = 12;
14         const unsigned REP = 100;
15
16         static_assert(sizeof(unsigned char) == 1, "sizeof(unsigned char) != 1");
17
18         std::vector<unsigned char> bits(SIZE);
19
20         std::cout << "#"
21                   << std::setw(GAP - 1) << "line"
22                   << std::setw(GAP    ) << "time"
23                   << std::endl;
24
25         for (unsigned line = 1; line <= LINE; line <<= 1)
26         {
27                 using namespace std::chrono;
28
29                 std::vector<double> sec(REP);
30                 for (auto &s: sec)
31                 {
32                         auto start = high_resolution_clock::now();
33                         for(unsigned i=0; i<line; i++){
34                                 for(unsigned j=0; j<SIZE; j+=line){
35                                         bits.at(j)++;
36                                 }
37                         }
38                         auto stop = high_resolution_clock::now();
39                         //s = duration_cast<nanoseconds>(stop - start).count() * line / double(bits.size());
40                         s = duration_cast<nanoseconds>(stop - start).count() / double(bits.size());
41                 }
42
43                 nth_element(sec.begin(), sec.begin() + sec.size() / 2, sec.end());
44                 std::cout << std::setw(GAP) << line
45                           << std::setw(GAP) << std::fixed << std::setprecision(3)
46                           << std::setw(GAP) << sec[sec.size() / 2]
47                           << std::endl;
48         }
49 }
```

En esta figura anterior encontramos el código del programa size.cc, el cuál solo he modificado el

bucle intermedio para que realice un nuevo bucle (este anidado al principal for), para que realice el microprocesador operaciones de escritura, ya que estas operaciones son más “lentas” que las lecturas, y podamos calcular mejor los tiempos del programa.