

Documentación

PRÁCTICA 6

Estructura de computadores



**UNIVERSIDAD
DE GRANADA**

Fernández Mertanen, Jonathan

lscpu

```
1: jonathan@jonathan-K55A: ~  
jonathan@jonathan-K55A:~$ 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: 2  
Núcleo(s) por «socket»: 2  
«Socket(s)»: 1  
Modo(s) NUMA: 1  
ID de fabricante: GenuineIntel  
Familia de CPU: 6  
Modelo: 58  
Nombre del modelo: Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz  
Revisión: 9  
CPU MHz: 1197.410  
CPU MHz máx.: 3200,0000  
CPU MHz mín.: 1200,0000  
BogoMIPS: 5188.56  
Virtualización: VT-x  
Caché L1d: 32K  
Caché L1i: 32K  
Caché L2: 256K  
Caché L3: 3072K  
CPU(s) del nodo NUMA 0: 0-3  
Indicadores: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat p  
se36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm constant_tsc arch_perfmon pebs  
_bts rep_good nopl xtopology nonstop_tsc cpuid aperfperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tm  
2 ssse3 cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer aes xsave avx f16c rdrand lah  
f_lm cpuid_fault epb pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid fsgsbase smep erms x  
saveopt dtherm ida arat pln pts flush_l1d  
jonathan@jonathan-K55A:~$
```

cpu-world

Cache details				
Cache:	L1 data	L1 instruction	L2	L3
Size:	2 x 32 KB	2 x 32 KB	2 x 256 KB	3 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

line.cc

Código

```
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] += 1;

            auto stop = high_resolution_clock::now();

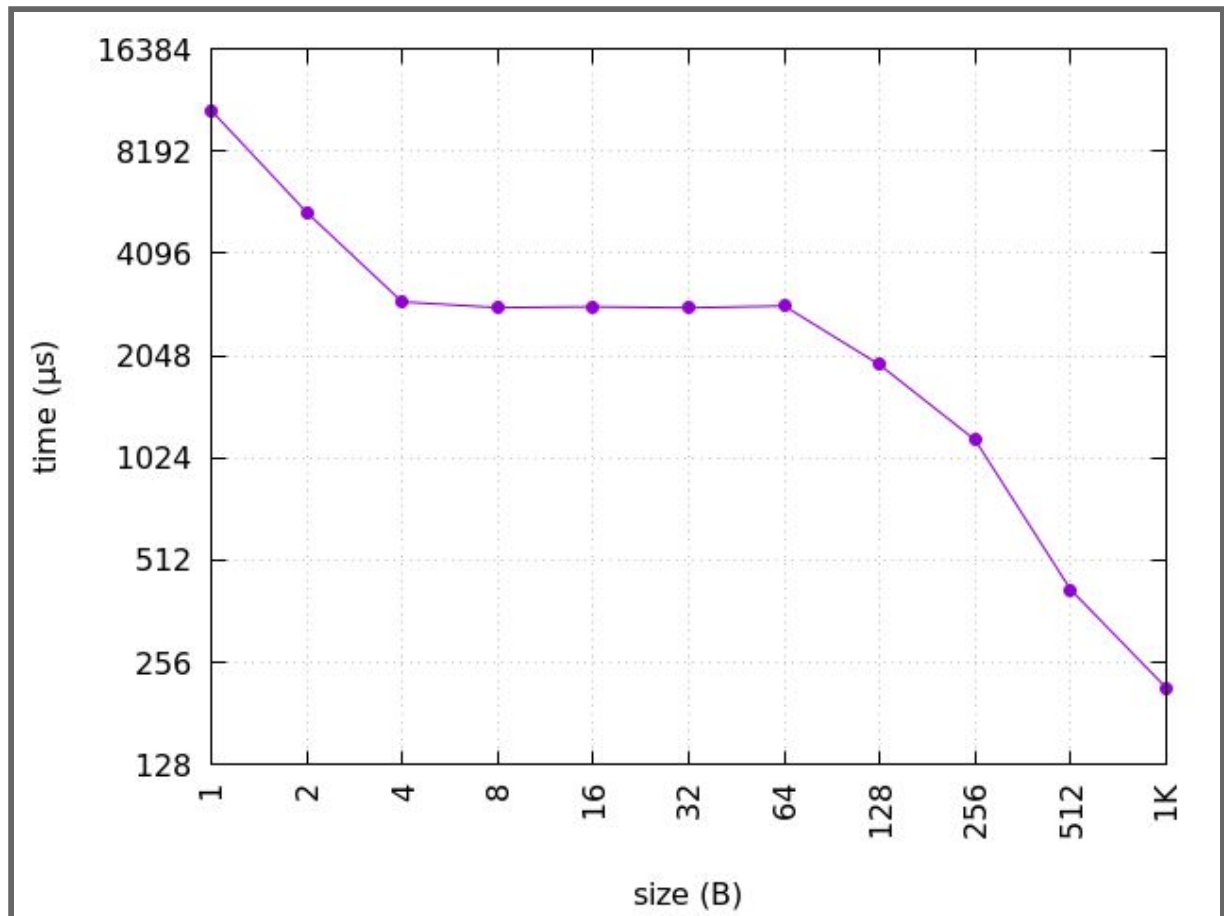
            s = stop - start;
        }

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

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

```

Gráfica



Podemos observar cómo se produce un cambio drástico en la velocidad a partir de 64B que se estabiliza, debido a que el tamaño de línea de mi ordenador es precisamente 64B. De esta manera, por debajo de 64, el tiempo es muy similar ya que no se producen errores de caché.

size.cc

Código

```
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();

        for (unsigned i = 0; i < STEPS; ++i){
            bytes[i* (1<<6) & (size - 1)]++;
        }

        auto stop = high_resolution_clock::now();

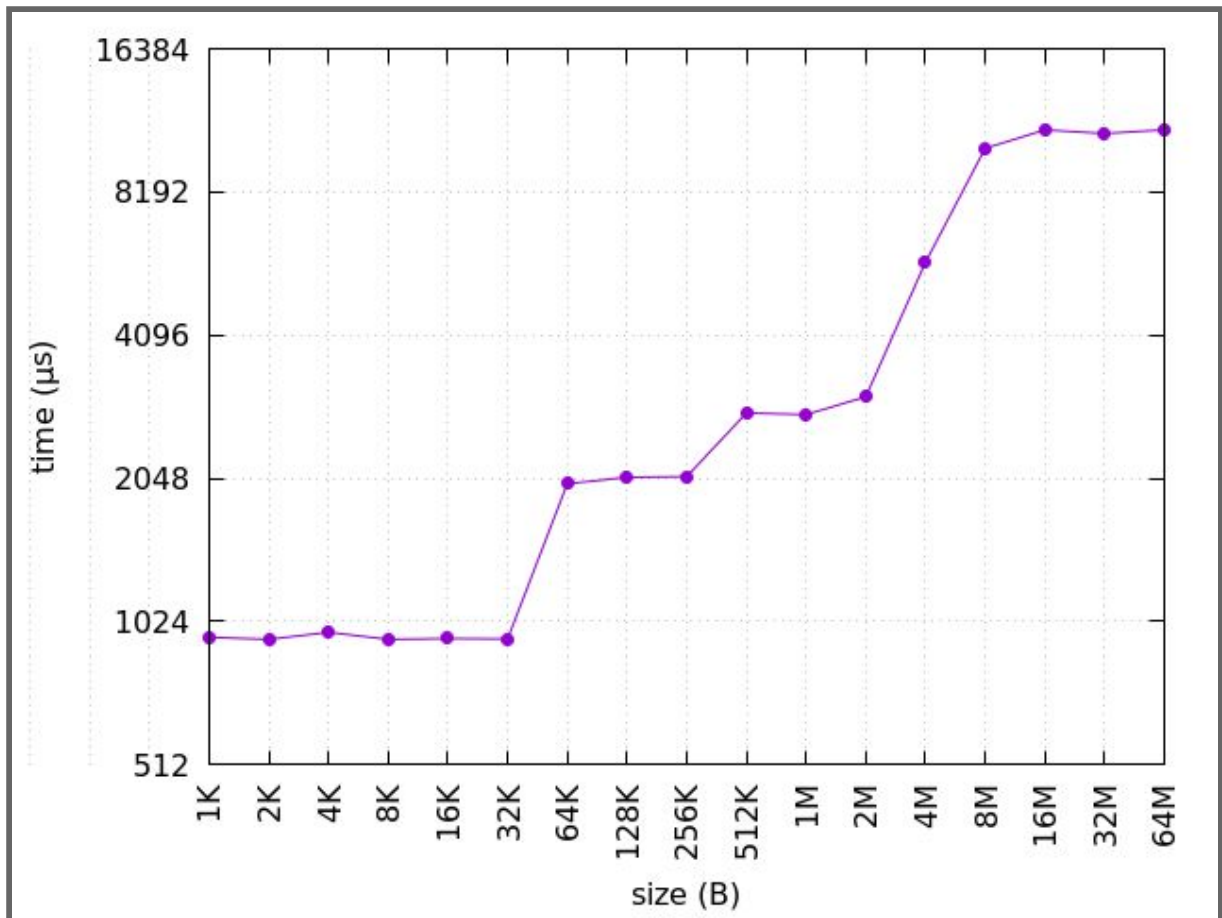
        s = stop - start;
    }

    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()
              << std::endl;
    }
}

```

Gráfica



En esta gráfica podemos observar los diferentes niveles de cache, que en mi caso, son 3:

- L1: 32K
- L2: 256K
- L3: 3M

y los saltos de tiempos que se producen en cada uno de ellos, siendo más notables los saltos de L1 y L3