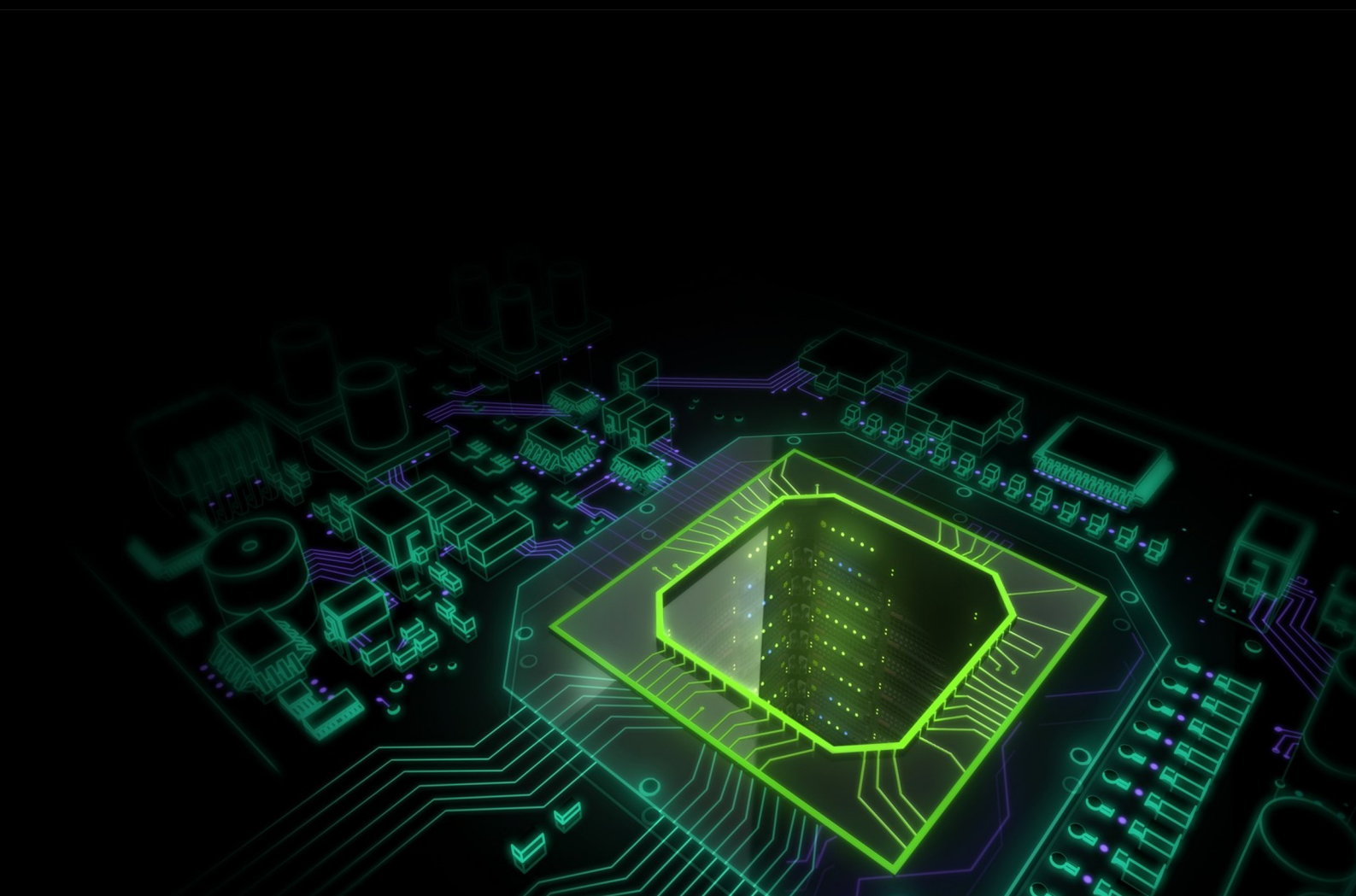
PRÁCTICA 5

Julio Fresneda

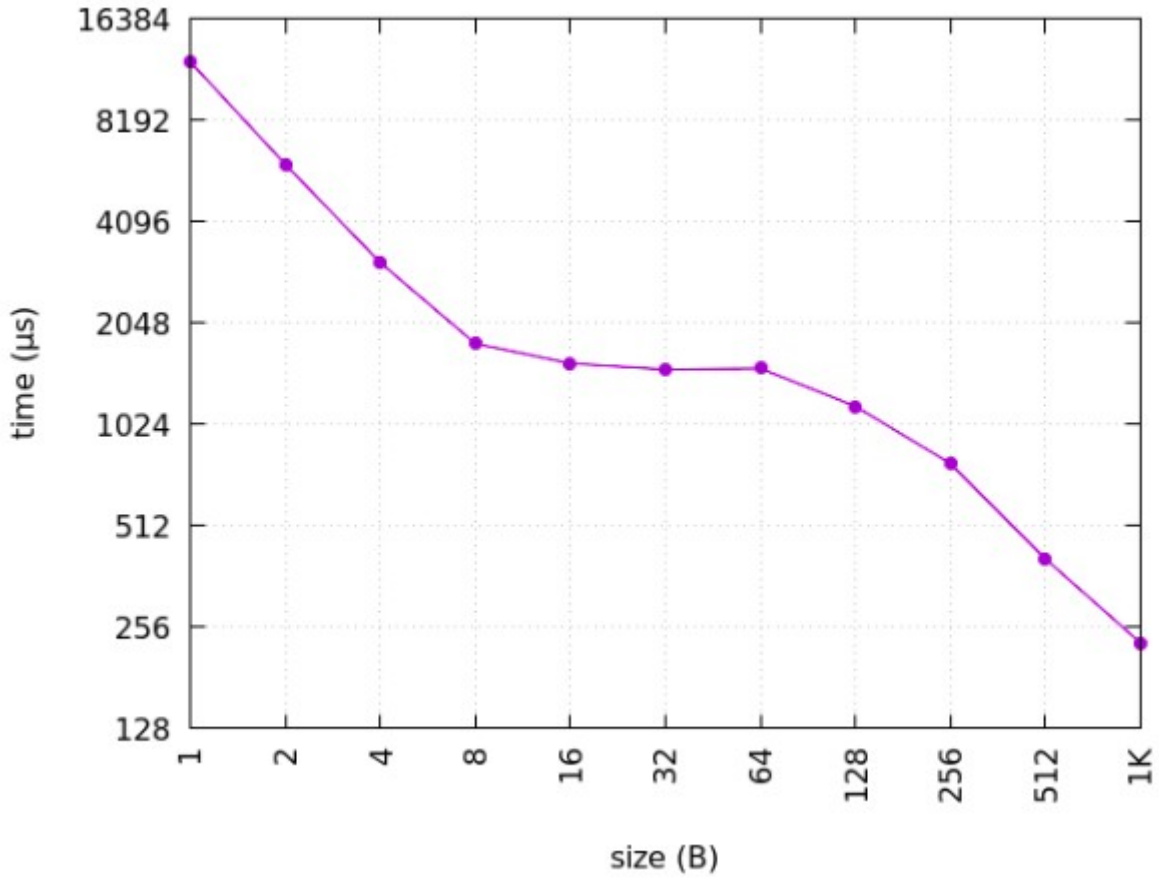
18-01-17



Archivo line.cc

```
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] ^= 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:

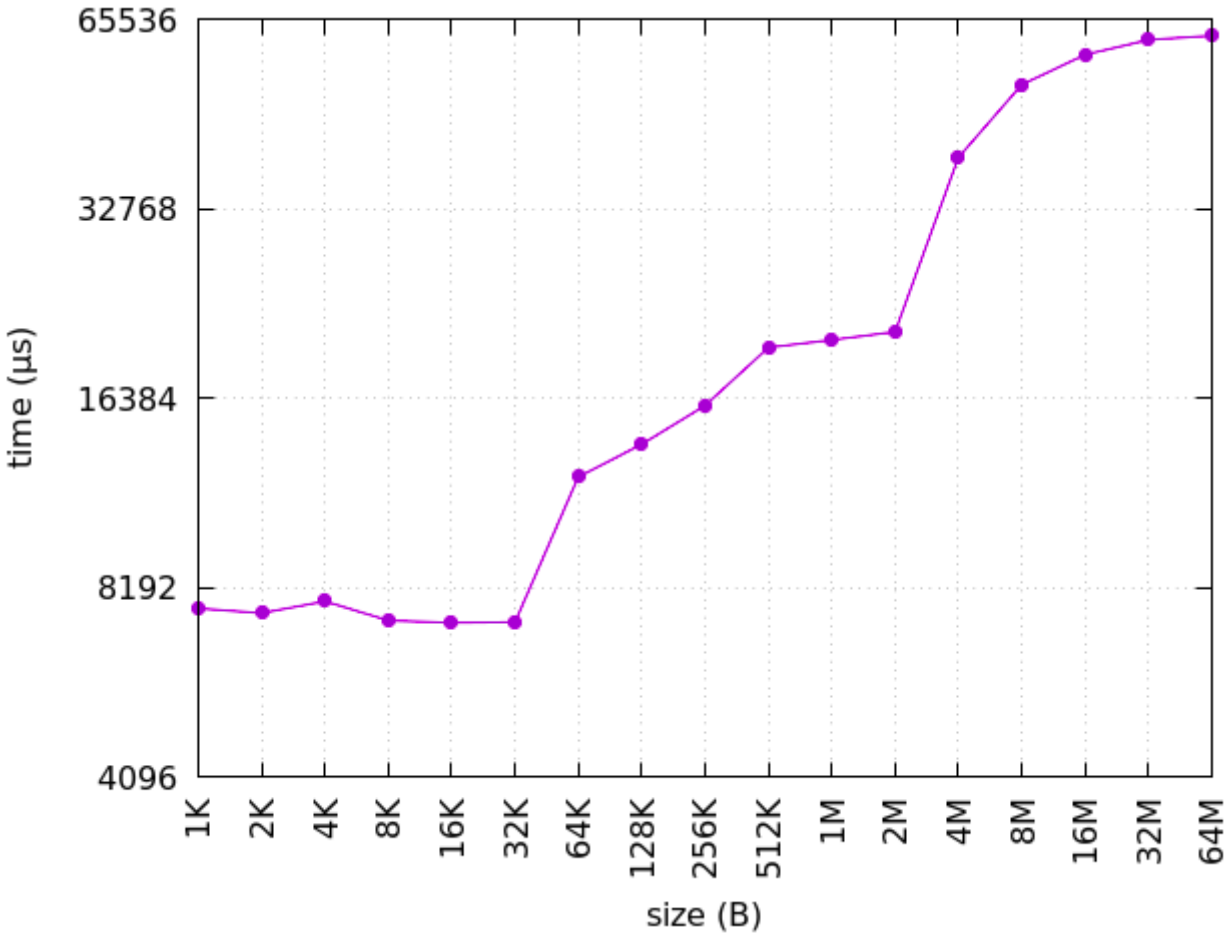


La gráfica es descendiente ya que, cuanto más grande es el tamaño de línea, más información puede pasar a la vez a través de ella a la vez, por lo que un traspaso de bytes tarda menos. Por eso al aumentar el tamaño de línea disminuye el tiempo.

Archivo size.cc

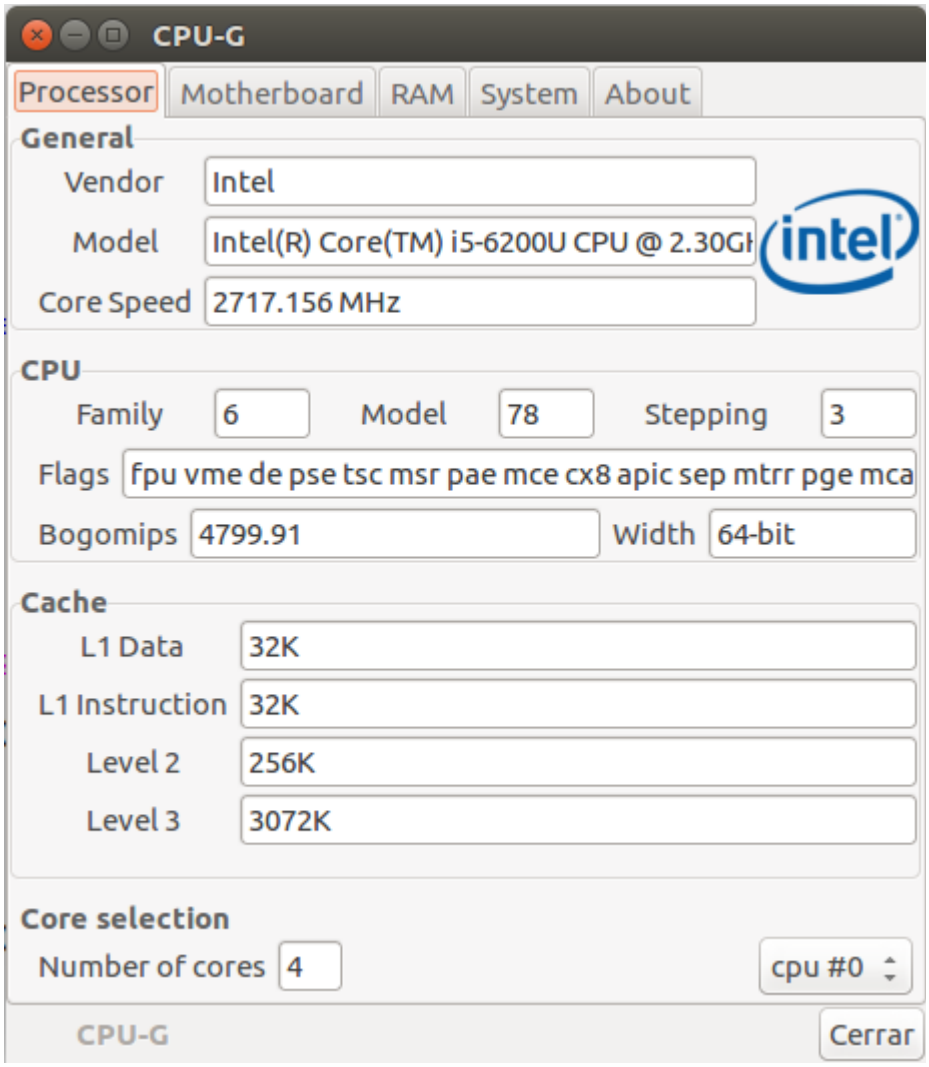
```cpp
using namespace std::chrono;

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 = 10;          // number of repetitions of every test
const unsigned STEPS = 1e7;        // 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();

                        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()
                          << std::endl;
        }
}
```

Gráfica



Mi procesador tiene tres niveles de caché. El nivel L1 tiene un tamaño de 32K. Por eso en la gráfica, al llegar a 32K, llena la caché y tiene que usar L2, que es mucho más lenta. Ésto se ve claramente en la gráfica, con una subida repentina. La subida de L2 y L3 no se ve tan clara, ya que usan tecnología muy parecida.

Características CPU



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	4-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