Práctica 5: Caché.

Vicente Martínez Rodríguez

Código de line.cc

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

int main()
{
        const unsigned LINE = 1 << 10;  // 1024B
        const unsigned SIZE = 1 << 21;  // 2MB
        const unsigned GAP = 12;
        const unsigned REP = 100;

        static_assert(sizeof(unsigned char) == 1, "sizeof(unsigned char) != 1");

        std::vector<unsigned char> bits(SIZE);

        std::cout << "#"
                << std::setw(GAP - 1) << "line"
                << std::setw(GAP    ) << "time"
                << std::endl;

        for (unsigned line = 1; line <= LINE; line <<= 1)
        {
                using namespace std::chrono;

                std::vector<double> sec(REP);
                for (auto &s: sec)
                {
                        auto start = high_resolution_clock::now();
```

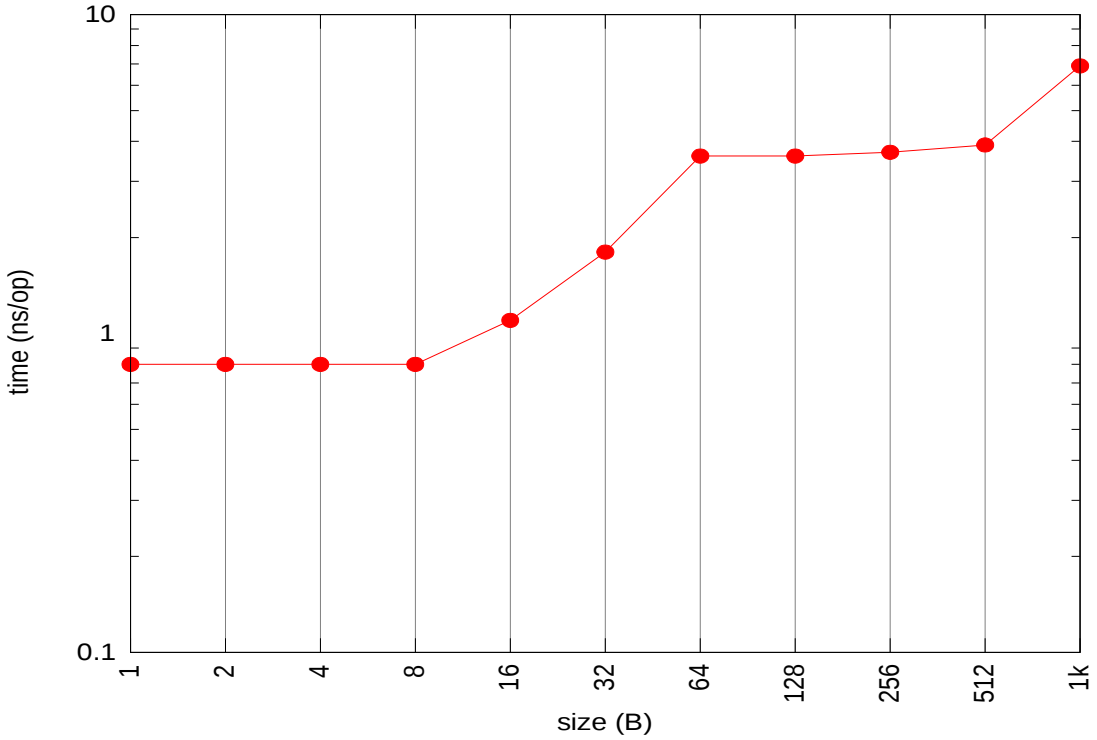
```
                        // completar aquí
                        // hacer una sola pasada y el tiempo que sale multiplicarlo por line
                        for(unsigned long long i=1; i<SIZE; i+=line)
                                bits[i]++;

                        auto stop = high_resolution_clock::now();
                        s = duration_cast<nanoseconds>(stop - start).count() * line /
double(bits.size());
                }

                nth_element(sec.begin(), sec.begin() + sec.size() / 2, sec.end());
                std::cout << std::setw(GAP) << line
                        << std::setw(GAP) << std::fixed << std::setprecision(1)
                        << std::setw(GAP) << sec[sec.size() / 2]
                        << std::endl;
        }
}
```

Código de size.cc

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

int main()
{
        const unsigned STEPS = 1000000;
        const unsigned GAP = 12;
        const unsigned REP = 32;
```

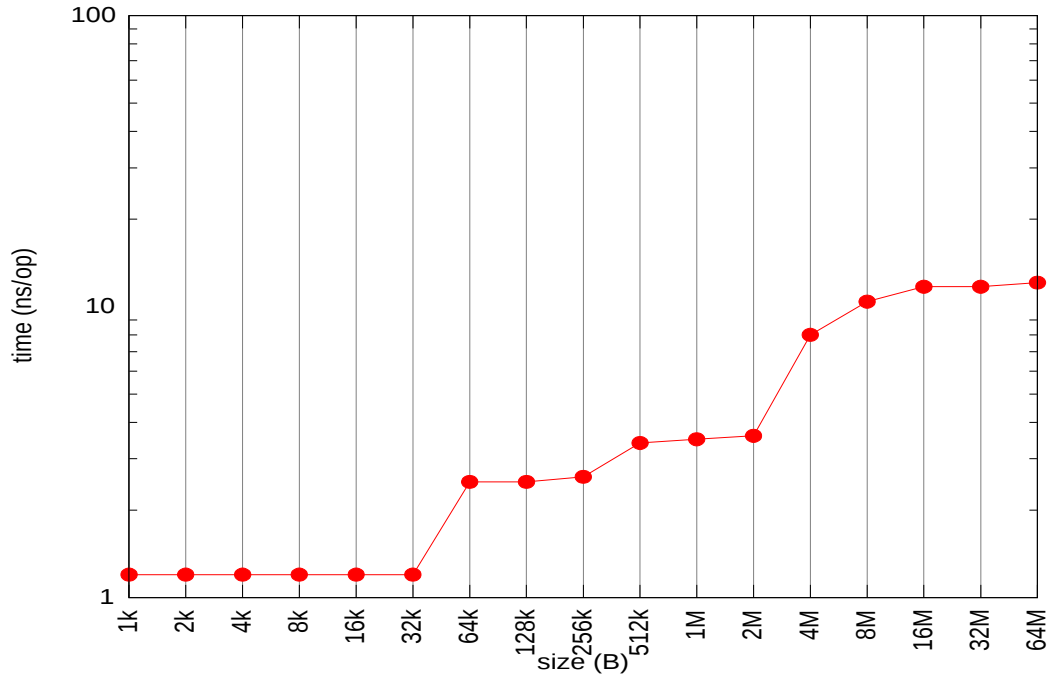
```
	std::cout << "#"
		<< std::setw(GAP - 1) << "size"
		<< std::setw(GAP    ) << "time"
		<< std::endl;
                                                        //size <= 1ULL << 33
	for (unsigned size = 1 << 10; size <= 1 << 26; size <<= 1)
	{
		using namespace std::chrono;

		std::vector<unsigned char> bits(size);

		std::vector<double> sec(REP);
		for (auto &s: sec)
		{
			auto start = high_resolution_clock::now();

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

			auto stop = high_resolution_clock::now();
			s = duration_cast<nanoseconds>(stop - start).count() / double(STEPS);
		}

		nth_element(sec.begin(), sec.begin() + sec.size() / 2, sec.end());
		std::cout << std::setw(GAP) << size
			<< std::setw(GAP) << std::fixed << std::setprecision(1)
			<< sec[sec.size() / 2]
			<< std::endl;
	}
}
```

Resultado de line:



El tamaño de línea es de 64 Bytes

Resultado de size:



Presenta 3 niveles. El primer nivel llega hasta 32Kb, el segundo es de 256KB y el tercero es de 3MB (la gráfica no representa impares).

-**Modelo de Procesador:**

Intel(R) Core(TM) i5-3317U CPU @ 1.70GHz

-**Level 1 cache size:**

2 x 32 KB instruction caches
2 x 32 KB data caches

-**Level 2 cache size:**

2 x 256 KB

-**Level 3 cache size:**

3 MB

http://www.cpu-world.com/CPUs/Core_i5/Intel-Core%20i5-3317U%20Mobile%20processor.html

Diario de trabajo:

Miércoles 8 de Enero de 2014: Se explica la práctica, realizo el bucle necesario para line.cc y obtengo la gráfica.

Miércoles 15 de Enero de 2014: Termino la práctica acabando size.cc y obteniendo la gráfica correspondiente, también busco en cpu-world las especificaciones de mi procesador.