Estructura de Computadores (2016-2017)
Subgrupo C3
Grado en Ingeniería Informática
Universidad de Granada

Práctica 5: Caché

Mario Rodríguez Ruiz

17 de enero de 2017

Índice

Índice de figuras

1. Procesador de las pruebas

Se trata del procesador **Intel Pentium CPU 2020M de 2.40GHz**.
En la página de **Cpu-World** aparece muy bien detalladas sus características (http://www.cpu-world.com/CPUs/Pentium_Dual-Core/Intel-Pentium%20Mobile%202020M.html).

La Figura 1.1 muestra la información básica de éste.

General information

Type	CPU / Microprocessor
Market segment	Mobile
Family	Intel Pentium Dual-Core Mobile
Model number	2020M
CPU part numbers	AW8063801033501 is an OEM/tray microprocessor AW8063801211202 is an OEM/tray microprocessor AW8063801539300 is an OEM/tray microprocessor
Frequency	**2400 MHz**
Bus speed	**5 GT/s DMI**
Clock multiplier	**24**
Package	988-pin micro-FCPGA10 (rPGA988B)
Socket	Socket G2 / rPGA988B
Size	1.48" x 1.48" / 3.75cm x 3.75cm
Introduction date	September 30, 2012
End-of-Life date	Last order date for tray processors is December 5, 2014 Last shipment date for tray processors is June 5, 2015

Figura 1.1: Primer indicio para averiguar la contraseña

Es ya en la Figura 1.2 donde aparece la información que interesa en este estudio. En ella puede los valores para los distintos niveles de caché, así como sus características.

Architecture / Microarchitecture

Microarchitecture	Ivy Bridge
Processor core	Ivy Bridge
Core steppings	E1 (SR184) L1 (QCH7, SR0U1) P0 (SR0VN)
CPUID	306A9 (SR0VN)
Manufacturing process	0.022 micron
Data width	**64 bit**
The number of CPU cores	**2**
The number of threads	**2**
Floating Point Unit	Integrated
Level 1 cache size	2 x 32 KB 8-way set associative instruction caches 2 x 32 KB 8-way set associative data caches
Level 2 cache size	**2 x 256 KB 8-way set associative caches**
Level 3 cache size	**2 MB 8-way set associative shared cache**
Physical memory	32 GB

Figura 1.2: Primer indicio para averiguar la contraseña

Otra forma de ver las características del cpu, pero esta vez desde la propia máquina, es mediante el comando **lscpu**. La ejecución de éste en este estudio puede verse en la Figura 1.3.

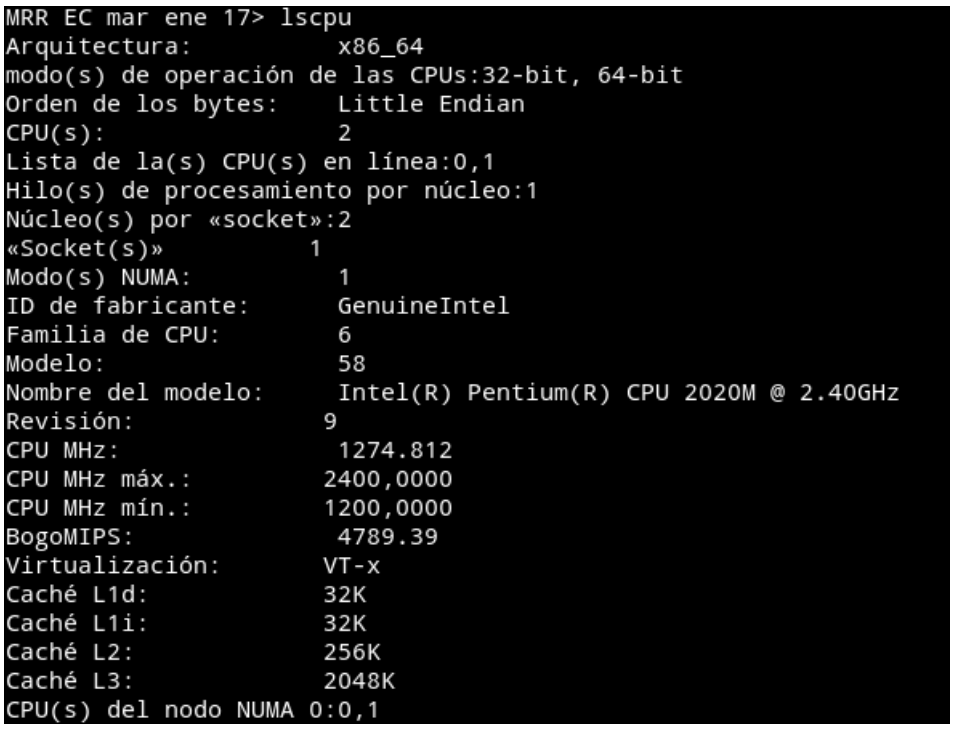

Figura 1.3: Primer indicio para averiguar la contraseña

Una manera adicional de obtener las características de la caché desde la propia máquina es la que se ha proporcionado en el **Makefile** de esta misma práctica. Su ejecución se muestra en la Figura 1.4

Figura 1.4: Primer indicio para averiguar la contraseña

2. Tamaño de linea

La parte del código **line.cc** que se ha modificado ha sido la siguiente:

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

Puede verse ésta en el código completo de a continuación:

```
// Mario Rodríguez Ruiz
#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 (ps)"
            << 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 long long j = 0; j < line; j++)
                for (unsigned i = 0; i < bytes.size(); i +=
                   line)
                  bytes[i]++;
```

```
32
33        auto stop = high_resolution_clock::now();
34        s = stop - start ;
35      }
36
37      std::nth_element(score.begin(),
38                       score.begin() + score.size() / 2,
39                       score.end());
40
41      std::cout << std::setw(GAP) << line
42                << std::setw(GAP) << std::fixed << std::
                     setprecision(1)
43                << std::setw(GAP) << score[score.size() / 2].
                     count()
44                << std::endl;
45    }
46  }
```

../v1/line.cc

Después de la ejecución del programa **line** se ha obtenido, a través de **gnuplot**, la gráfica de la Figura 2.1.

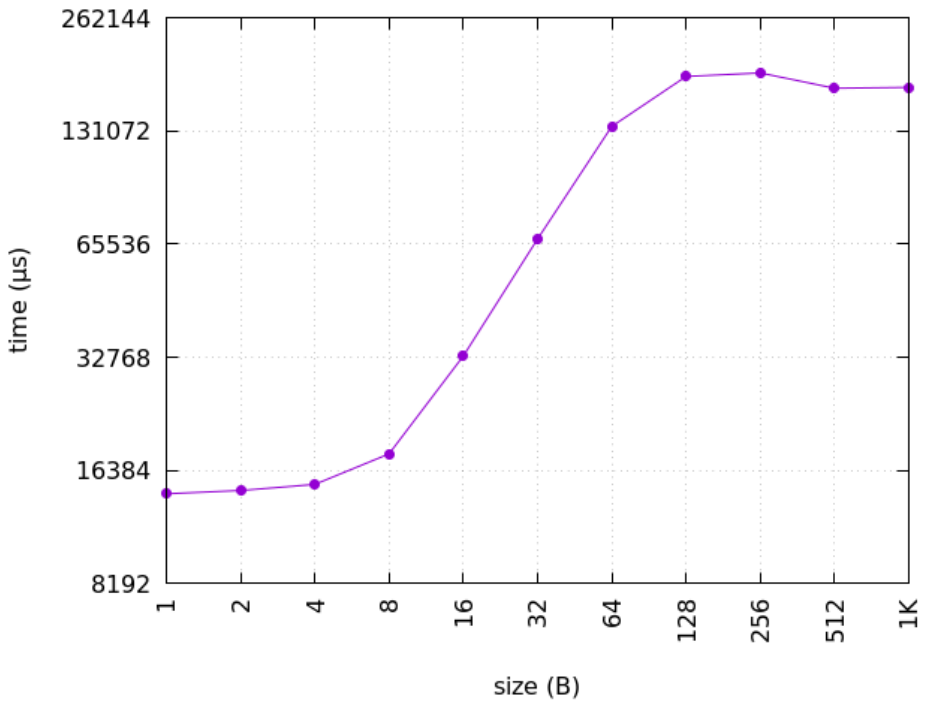

Figura 2.1: Resultado del tamaño de linea del procesador

En la gráfica puede verse el **incremento** de tiempo que hay desde **1B hasta 64B**, esto se produce porque se obtienen muchos fallos de caché ya que se acceden a datos que no se encuentran en ella. Es por ello por lo que se tiene que tener en cuenta solo el bloque que los contiene de memoria principal.

Justo después, los resultados se estabilizan cuando tenemos bastantes aciertos en caché. El pico donde se empieza a estabilizar es lo que se conoce como el **tamaño de línea de caché**, que en este caso es de **64B**.

3. Tamaño de caché

La parte del código **size.cc** que se ha modificado ha sido la siguiente:

```
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;
```

Puede verse ésta en el código completo de a continuación:

```
// Mario Rodríguez Ruiz
#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

```

```
17 int main()
18 {
19   std::cout << "#"
20             << std::setw(GAP - 1) << "line (B)"
21             << std::setw(GAP    ) << "time (ps)"
22             << std::endl;
23
24   for (unsigned size = MINSIZE; size <= MAXSIZE; size *= 2)
25   {
26     std::vector<duration<double, std::micro>> score(REP);
27
28     for (auto &s: score)
29     {
30       std::vector<char> bytes(size);
31
32       auto start = high_resolution_clock::now();
33
34       for (unsigned i = 0; i < STEPS; ++i)
35         bytes[(i*64)&(size-1)]++;
36
37       auto stop = high_resolution_clock::now();
38
39       s = stop - start;
40     }
41
42     std::nth_element(score.begin(),
43                      score.begin() + score.size() / 2,
44                      score.end());
45
46     std::cout << std::setw(GAP) << size
47               << std::setw(GAP) << std::fixed << std::
                     setprecision(1)
48               << std::setw(GAP) << score[score.size() / 2].
                     count()
49               << std::endl;
50   }
51 }
```

../v1/size.cc

Después de la ejecución del programa **size** se ha obtenido, a través de **gnuplot**, la gráfica de la Figura 3.1.

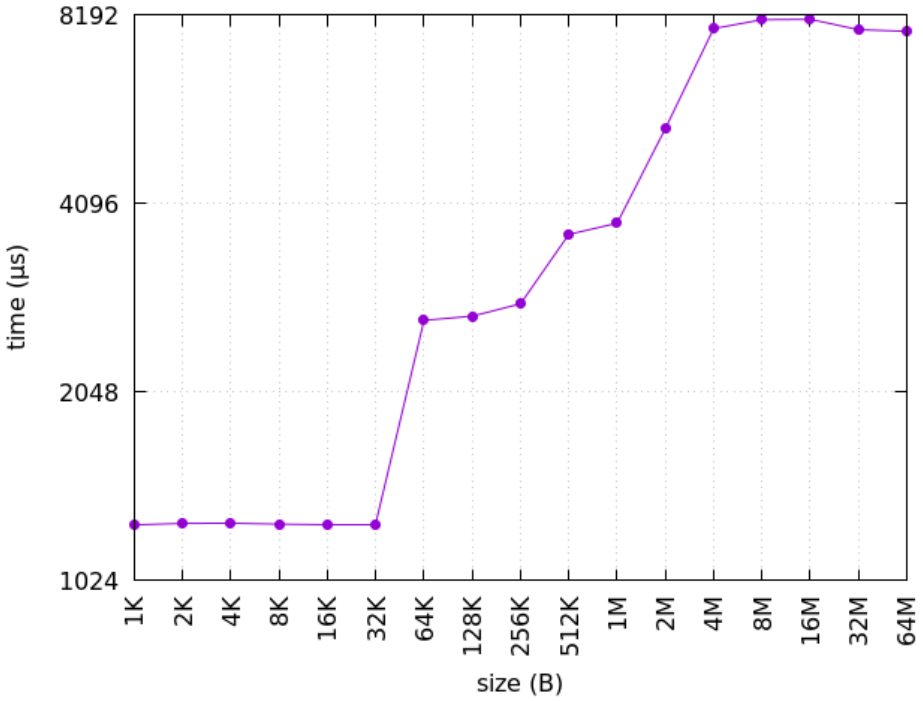

Figura 3.1: Resultado del tamaño de caché del procesador

En la gráfica anterior puede observarse cómo se mantiene **estable** el tiempo **desde 1K hasta 32K**. Justo después se tiene un incremento de tiempo significativo, producido porque se ha ocupado todo el espacio de la **caché L1** y ya se está accediendo a la **L2**. Ese incremento en el tiempo (produciendo un pico) indica que el **tamaño de la caché L1 es de 32KB**.

El tiempo desde **64K hasta 256K es estable**, pero justo a continuación se obtiene un incremento de tiempo importante que indica que la L2 se ha ocupado por completo y se empezará a usar la L3. El pico de subida creado por este incremento muestra que el **tamaño de la caché L2 es de 256K**.

Posteriormente, **desde 512k hasta poco antes de 2M** se obtienen resultados estables de nuevo. La conclusión de esta estabilidad indica que la cache L3 se ha ocupado por completo y, a partir de este momento, los accesos se realizarán sobre memoria principal. Es por ello por lo que se puede afirmar que la caché **L3 es de 2M**.

La presentación de estos datos puede ayudar a entender la importancia que tienen tamaños de caches que, cuanto más lejos del procesador más tiempo existe de acceso a datos. Es por ello por lo que la L1 es la más rápida y la memoria principal la más lenta.