Práctica AC

Práctica 2

Evaluación del Rendimiento

Grado en ingeniería informática

Jesús López Galbis 15422155B
Raúl Beltrán Marco 23900664F
Óscar David Tremiño Guirao 74386585P
Francisco Joaquín Murcia Gómez 48734281H
Yana Spiridonova Y2784928X
Pedro López Mellado 78857478T

ÍNDICE

Introducción

En esta práctica desarrollaremos e implementaremos un benchmark reducido, el cual nos ayudará a determinar el rendimiento de los computadores y compararlos entre sí.

Además, observaremos cómo varía el tiempo de ejecución implementando diferentes mejoras en su implementación y discutiremos los resultados aportados por el programa.

Nuestro benchmark consistirá en un pequeño código que multiplicará una matriz de MxM tantas veces como deseemos, la base del algoritmo es la multiplicación de filas x columnas que se implementará en ensamblador.

Una vez realizado nuestro propio benchmark en nuestros equipos con sus diferentes implementaciones, vamos a comprobar y comparar los resultados obtenidos por nuestro propio benchmark con la herramienta SPEC cpu2000.

Implementación del Benchmark

Explicación del código

Nuestro código está formado por tres partes, la generación de matrices, la multiplicación entre ellas y la medición del tiempo.

Primero generamos un vector de vectores de MxM el cual representa a la matriz,

después, a la hora de rellenar la matriz recorremos el vector y con la función “rand()” de la librería “stdlib” asignamos los valores.

```
srand(time(NULL));//Semilla para randomizar

for (unsigned int i = 0; i < M; i++)
    for (unsigned int j = 0; j < M; j++)
        vector[i][j] = rand() % 20 + 1;
```

generación de matrices

Después multiplicamos la matriz por si misma para ahorrar memoria, esta parte es diferente dependiendo del lenguaje de ensamblador que usemos, el funcionamiento es idéntico al método de multiplicar matrices por “filas por columnas”, multiplicamos el elemento n de la fila por el elemento n de la columna y le sumamos la multiplicación de los elementos n+1.

```
int aux[M][M];
for (int i = 0; i < M; i++)
{
    for (int j = 0; j < M; j++)
    {
        aux[i][j] = 0;
        for (int k = 0; k < M - 1; k++)
        {
            aux[i][j] = aux[i][j] + vector[i][k] * vector[k][j];
        }
    }
}
```

filas por columnas en c++

Nuestro benchmark mide el tiempo en realizar la operación, para ello hemos usado una función que usa "QueryPerformanceFrequency()" que hemos sacado del foro "stackoverflow", hemos usado "QueryPerformanceFrequency()" porque es mucho más preciso que "clock()" de la librería "chrono" y a contiene más decimales, el único inconveniente es que esta función no se encuentra disponible en windows, para eso hay que añadir la libreria "windows.h", para más información, en la bibliografía.

Este proceso lo realizamos 500 veces para obtener un resultado más significativo, a parte, el programa realiza la prueba 10 veces para obtener una media para tener el resultado más aproximado a la realidad.

Implementación en C++

Para la implementación en C++ hemos realizado 3 bucles *for* anidados, con los cuales poder recorrer todos los elementos de la matriz que queremos multiplicar por sí misma.

La operación que realizamos y la que conlleva mayor coste para el ordenador es: aux[i][j] = aux[i][j] + vector[i][k] * vector[k][j]

En esta instrucción accedemos a 3 posiciones distintas de las matrices y realizamos la multiplicación haciendo uso del *operator+* y el *operator** los cuales nos ofrece el propio lenguaje de C++

```
long double start = clock();
int aux[M][M];
for (int i = 0; i < M; i++)
{
    for (int j = 0; j < M; j++)
    {
        aux[i][j] = 0;
        for (int k = 0; k < M - 1; k++)
        {
            aux[i][j] = aux[i][j] + vector[i][k] * vector[k][j];
        }
    }
}
long double end = clock();
tiempo += (end - start) / 1000;
```

parte correspondiente en c++

Implementación en x86

Para la implementación en ensamblador x86 hemos usado dos variables auxiliares “a” y “b” para facilitar el procedimiento, primero pasamos los números “a” y “b” a los registros “eax” y “ebx” respectivamente con la instrucción “MOV destino, fuente”. Después usamos la instrucción “MUL reg”, está multiplica el registro que le pasa por el registro “eax”, y lo almacena en “eax”, en este caso multiplicamos “ebx” (que es el parámetro). A continuación, pasamos el resultado con “MOV destino, fuente” de “eax” a la variable “a”. Por último, ponemos el resultado de “a” a la matriz resultado.

```
int aux[M][M];
for (int i = 0; i < M; i++)
{
    for (int j = 0; j < M; j++)
    {
        aux[i][j] = 0;
        for (int k = 0; k < M - 1; k++)
        {
            b = vector[i][k];
            a = vector[k][j];

            __asm
            {

                mov eax, a;
                mov ebx, b;
                mul ebx;
                mov a, eax;

            }

            aux[i][j] = aux[i][j] + a;
        }
    }
}
```

parte correspondiente en ensamblador x86

Implementación en SSE

Para la implementación en ensamblador SSE, hemos eliminado los tres bucles que recogen la matriz, a continuación, con la instrucción "movups" cogemos las matrices y los introducimos en los registros "xmm" correspondientes. Finalmente, con la instrucción "mulps" multiplicamos las dos matrices y almacenamos el resultado automáticamente en el registro "xmm0", finalmente, con "movups" pasamos el resultado del registro "xmm0", a la matriz "aux".

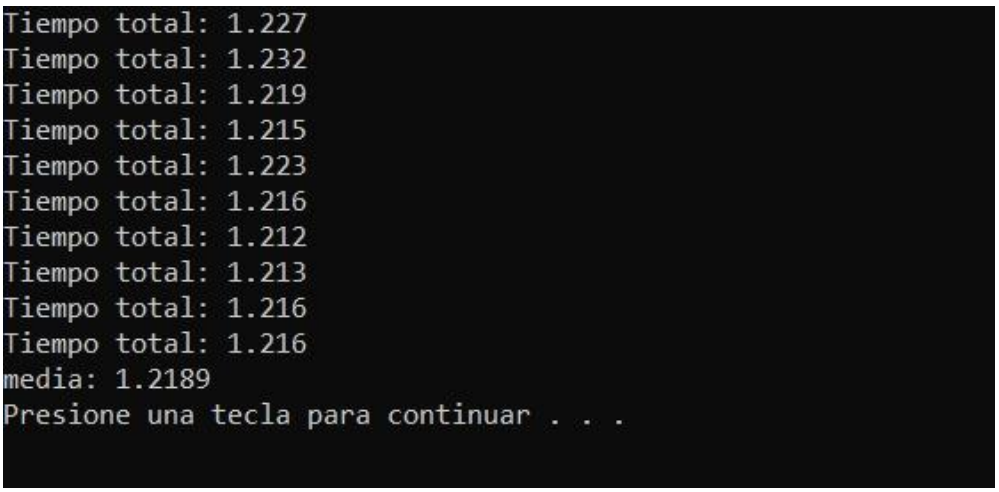
```
__asm
{

    movups xmm0, [vector];
    movups xmm1, [vector];
    mulps xmm0, xmm1;
    movups [aux], xmm0;

}
```

parte correspondiente a SSE

Evaluación de resultados

Interpretación de resultados

La forma que tiene nuestro programa de evaluar la eficiencia es midiendo el tiempo en multiplicar las 500 matrices M x M, de esta manera tenemos un resultado claro de la velocidad que el tiempo es inversamente proporcional a la velocidad, el programa nos muestra los tiempos de diez pruebas, de esta manera el valor de la media será el más aproximado al real.



salida por pantalla del benchmark

Equipos empleados

Estos son los equipos que hemos empleado para realizar las pruebas:

	Procesador	Nucleos	Frecuencia*	Turbo*	Nº hilos	AC*	SO	Opciones
Equipo 1	i5-8250U	4	1,60 GHz	3,40 GHz	8	3D	W10	portatil
Equipo 2	i7 8750H	6	2,20 GHz	4,10 GHz	12	3D	W10	portatil
Equipo 3	ryzen7 2700X	8	3.7G Hz	4.3GHz	16	2D	W10	torre
Equipo 4	i7 7700HQ	4	2, 80 GHz	3,8 GHz	8	3D	W10	portatil
Equipo 5	i7 8750H	6	2,20 GHz	4,10 GHz	12	3D	W10	torre
Equipo 6	Fx 6300	6	3.5GHz	3.8GHz	6	2D	W10	torre

***frecuencia: frecuencia base.**

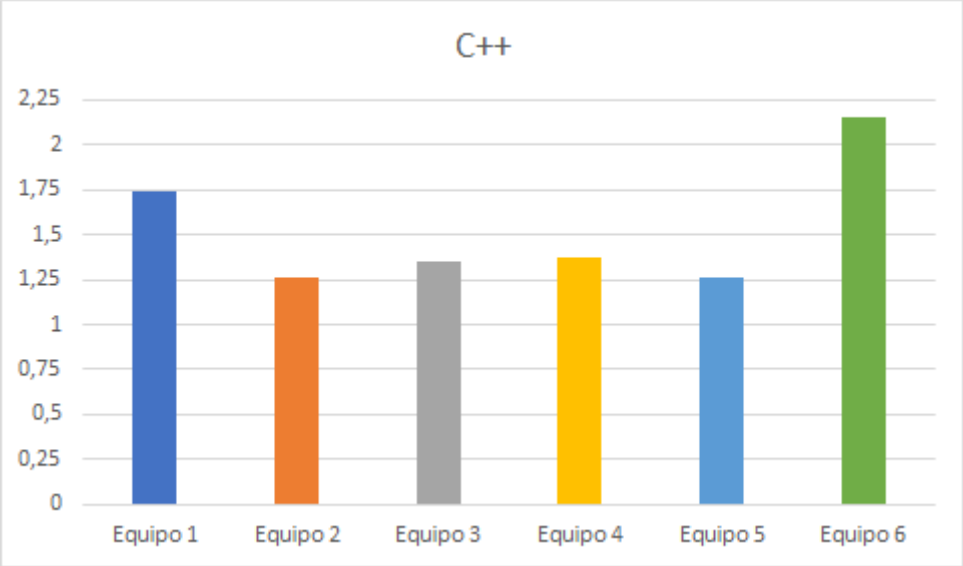
***Turbo: frecuencia turbo.**

***AC: arquitectura del procesador.**

Comparación de diferentes lenguajes

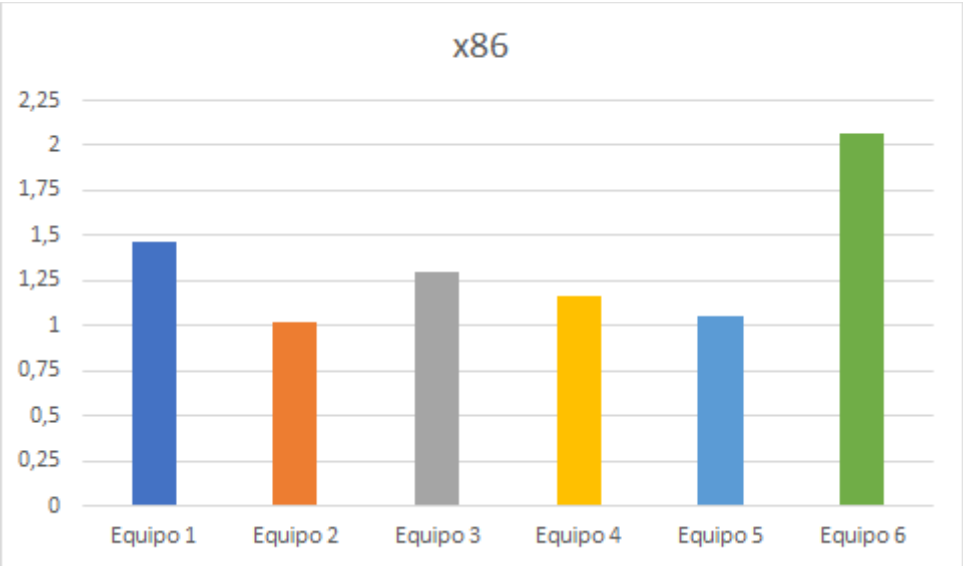
En este apartado compararemos el rendimiento de los diferentes lenguajes por cada equipo con una matriz 100*100.

A continuación, vamos a hacer una prueba en c++ con el código comentado anteriormente sobre los diferentes equipos de los integrantes del grupo:



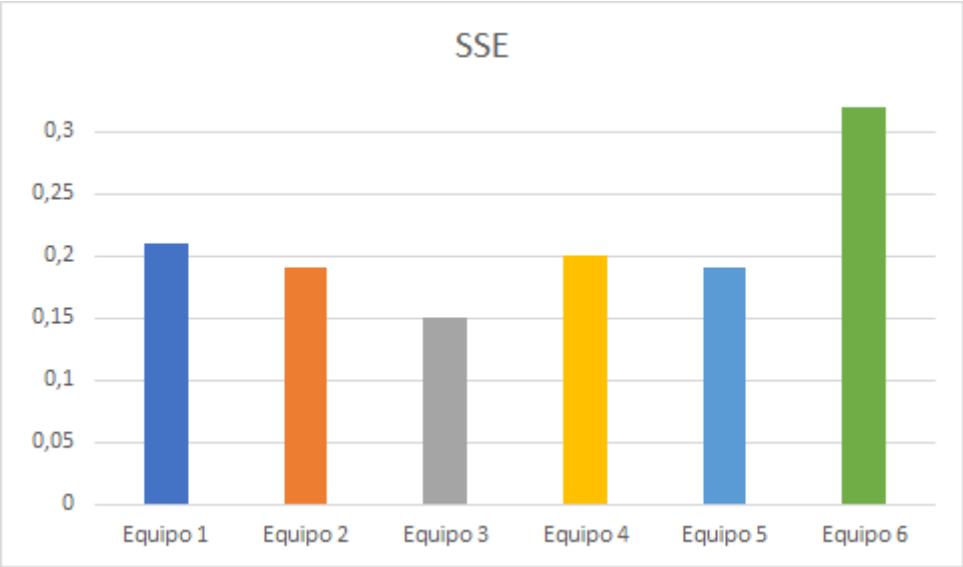
Una vez realizada la prueba podemos observar que el equipo más rápido de todos sería el i7-8750H, pero en cuanto a ello exceptuando el i5-8250U y el Fx-6300 los tiempos y velocidades de estos son muy parejas.

A continuación, vamos a hacer una prueba utilizando para la realización de la multiplicación el lenguaje ensamblador x86 de forma inline en el código en c++ de la misma forma en todos los equipos de los diferentes integrantes del grupo.

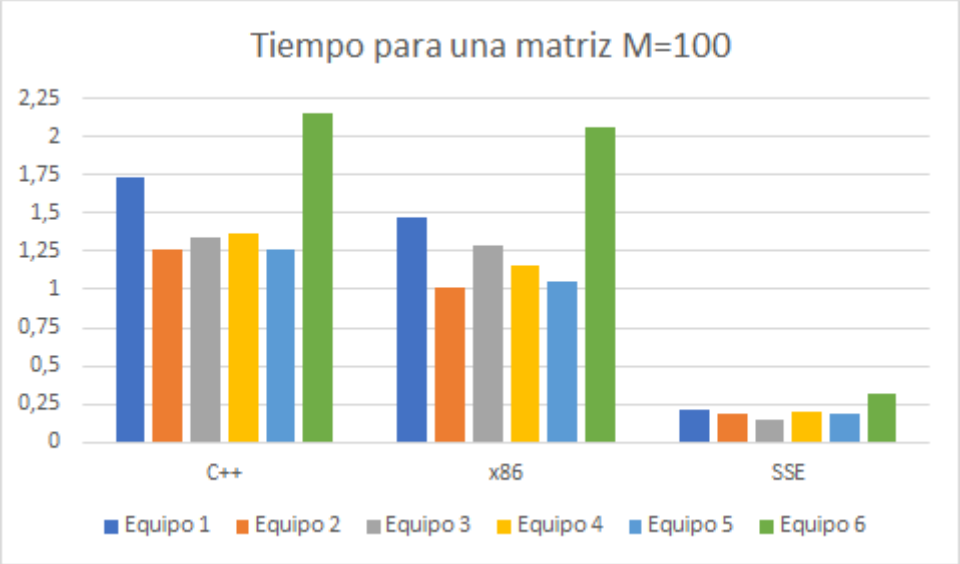


Una vez realizada la prueba podemos observar que los resultados son muy similares a los obtenidos en el caso anterior, se puede observar una tasa de mejora superior en los procesadores de la marca en los equipos con procesador de tipo intel que en los equipos de tipo AMD.

Finalmente vamos a hacer una prueba cambiando de nuevo la forma en la que se realiza la multiplicación de la matriz, ésta vez se realizará con código SSE inline de nuevo en c++, utilizando así mismo la paralización, los resultados que obtenemos son los siguientes:



Tras la realización de la prueba podemos observar que en este caso los tiempos se han reducido de forma sorprendente debido a la paralización y en este caso se puede observar una mejor respuesta por parte de los procesadores de tipo AMD sobre todo del Ryzen 7 2700X ya que este procesador consta de 8 núcleos con 16 hilos, independiente de que el equipo 6 consta de un procesador de la marca AMD, al ser un procesador con una antigüedad aproximada de 6 años teniendo en cuenta la velocidad a la que avanzan las nuevas tecnologías pues este se quedaría un tanto atrás.



A continuación en esta gráfica que muestra los resultados generales de todos los equipos, al pasarle las pruebas observamos que respecto a las dos primeras pruebas (las que pasamos con c++ y Ensamblador) el equipo 3 sería el equipo más veloz muy seguido del equipo 5 el cual tiene el mismo procesador pero es un ordenador de sobremesa(torre), este aspecto prácticamente no influye en el resultado final de los tiempos y velocidades a la hora de pasar el benchmark.

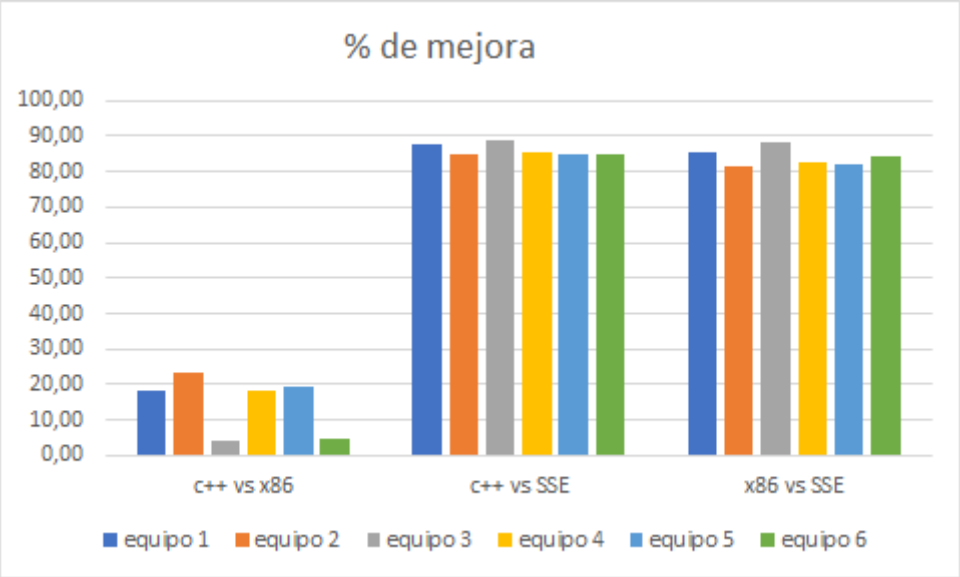
Por otra parte, el equipo 4 al considerarse un procesador enfocado al ahorro de energía sufre una pérdida de rendimiento respecto a sus homólogos que serían el equipo 2 y el equipo 5.

Tasa de mejora

En esta última gráfica se puede observar la tasa de mejora del rendimiento de los ordenadores a la hora de usar diferentes benchmark.

Se observa claramente que entre c++ y x86 a pesar de no existir una gran diferencia, pero aun así es notable, con una mejora de alrededor del 20% para los procesadores con filosofía de Intel y con una mejora de alrededor de un 5% para los de la filosofía AMD.

En cuanto al paso de c++ o x86 hacía SSE se observa una tasa de mejora alrededor del 85% tanto para los equipos 1, 2, 4, 5 y 6. Así mismo se puede observar que el equipo 3 alcanzan casi el 90% de mejora en el rendimiento, esto es debido a que posee el mayor número de núcleos.



Uso del paquete de evaluación SPEC

¿Qué es SPEC cpu2000?

Desarrollado por SPEC; un consorcio de fabricantes, universidades, centros de investigación, etc; SPEC cpu2000 es un paquete de benchmarks, contiene a su vez dos paquetes, el CINT2000 (pruebas con enteros) y el CFP2000 (pruebas con flotantes) cada paquete consta de 11 pruebas y 14 respectivamente, estos son pruebas como: Modelo de acelerador de partículas, Programa de ajedrez, Simulador de ubicación y ruteo, Química computacional, etc.

Cómo usar SPEC cpu2000

Una vez descargado SPEC cpu2000, hay que moverlo al disco "C", una vez allí, editamos el archiva "shrc.bat" para colocar nuestra dirección de donde vamos a compilar, en nuestro caso Visual Studio), se coloca en los "call" como se ve en la imagen:

```
rem the line that follows (just remove the word `rem`).
set SHRC_COMPILER_PATH_SET=yes
call "C:\Program Files (x86)\Microsoft Visual Studio\2019\Enterprise\VC\Auxiliary\Build\vcvars32.bat"
rem
rem If someone else has compiled the benchmarks for you, then the
```

Una vez hecho esto, con la consola de comandos entramos en el directorio de SPEC cpu2000 e ejecutamos los siguientes comandos:

- **find -name desktop.ini -delete**
  este elimina archivos innecesarios que pueden interferir en la prueba
- **./ shrc.bat**
  este comando compila los archivos
- **runspec --reportable --config=win32-x86-vc7.cfg -T base int**
  este comando ejecuta la prueba

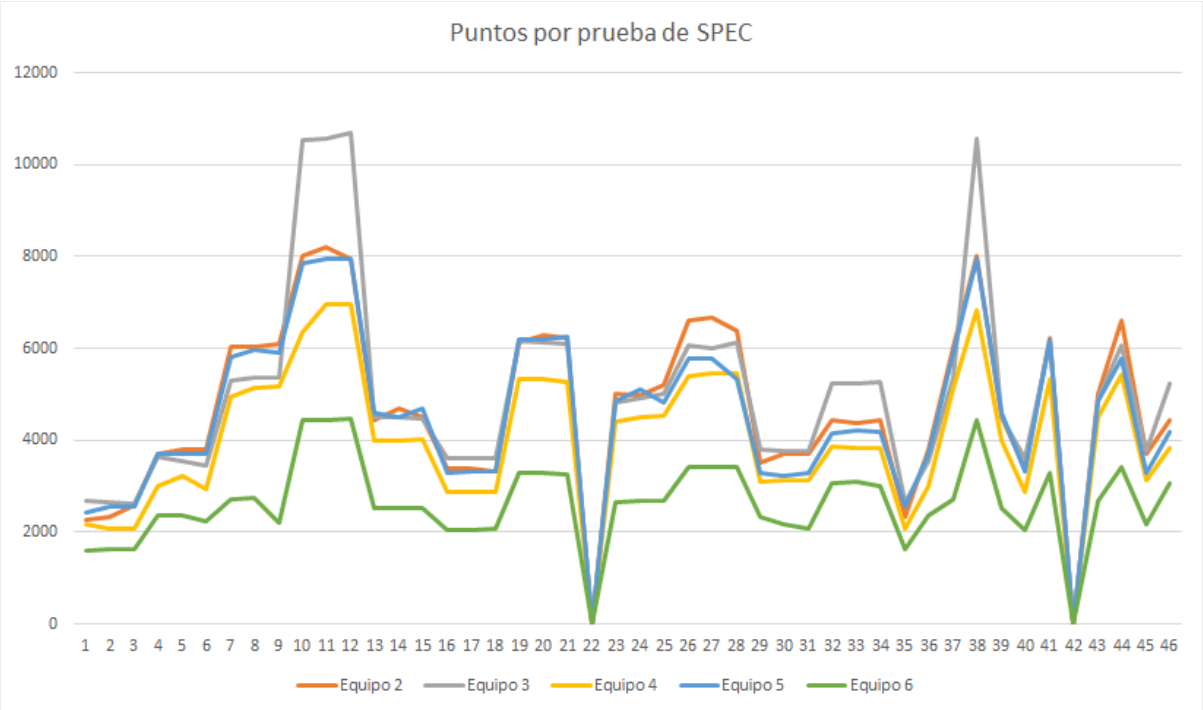
Una vez terminada la ejecución, en la carpeta "result" se encuentran los resultados de la prueba.

Análisis de resultados

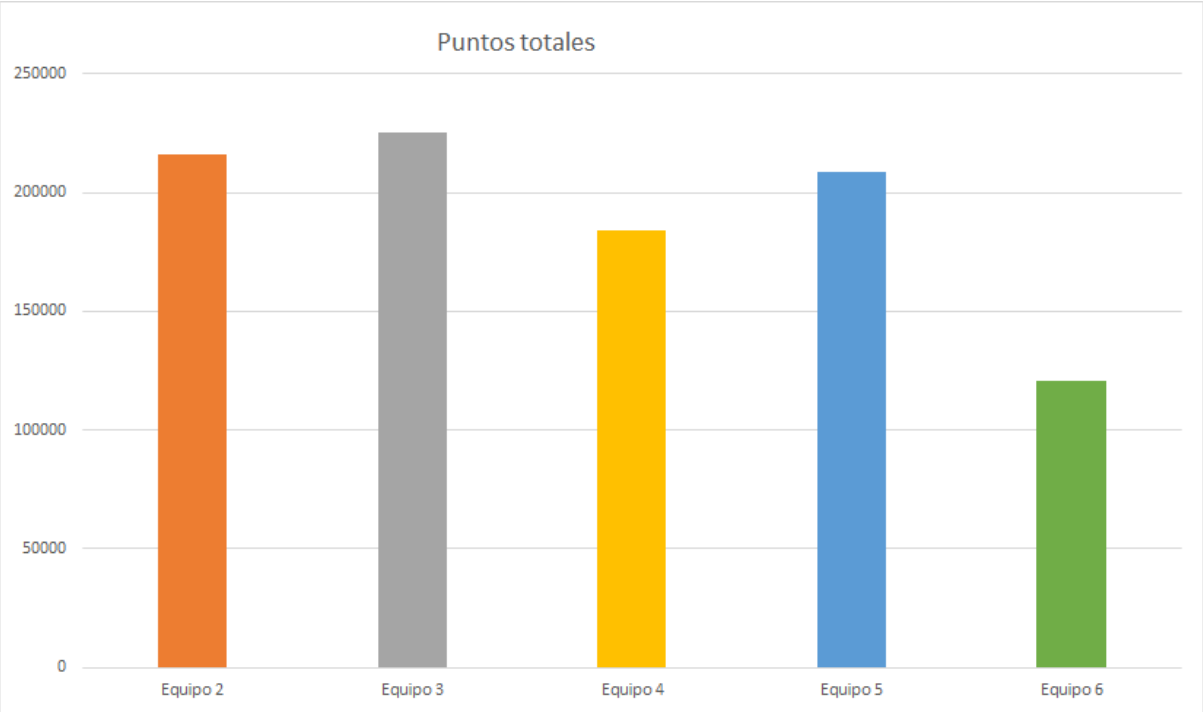
SPEC muestras sus resultados con un sistema de puntos, estos son proporcional con el rendimiento.

Al realizar la prueba en todos los equipos, el equipo 1 al ejecutar "./shrc.bat" se le congelaba la consola y no se pudo realizar la prueba, ese es el motivo por el que no aparece en las estadísticas de los resultados.

Los resultados son los siguientes:



Como podemos observar el equipo 3 destaca en las pruebas donde se paraleliza más, mientras que el equipo 2 y 5 se mantienen iguales la mayoría del tiempo, destacando en las pruebas que no se paraleliza, el equipo 4 al ser similar al 2 y 5 se mantiene un poco detrás de estos, en las pruebas que la puntuación nos da 0, son pruebas que no se han podido ejecutar.



Si hacemos un cómputo total de los puntos encontramos que el equipo 3 es el equipo con mayor rendimiento seguido muy de cerca por los equipos 2 y 3, es notable la falta de rendimiento del equipo 4 por ser un procesador de ahorro de energía, por último, con una gran diferencia, el equipo 6.

Conclusiones

Para concluir volvemos a hacer hincapié en los resultados obtenidos en nuestro benchmark y el de SPEC, como hemos podido observar tras la realización de las pruebas, los dos benchmarks han llegado a resultados muy parecidos dando como mejor equipo al número 3 (AMD Ryzen 7).

Aunque, tras varias pruebas hemos observado que no podemos determinar del todo bien la potencia del ordenador respecto de la tarea que le proponemos, ya que el ordenador siempre tiene otros procesos en segundo plano como E/S, actualizaciones de window… y cuantos más procesos tengamos abiertos menos recursos destinará a nuestra tarea en concreto.

Además, cabe destacar como hemos observado una notable mejora en nuestro benchmark reducido al implementarlo con diferentes lenguajes, cabe destacar la gran mejoría cuando se aprovechan los núcleos de los procesadores, llegando a ahorrar incluso un 90% en tiempo.

Las pruebas se han realizado tanto en ordenadores de sobremesa como en portátiles y cabe observar que en estos benchmarks reducidos, no se nota la diferencia.

Por último, nos gustaría destacar el avance de esta rama, ya que uno de los equipos (equipo 6) posee un procesador de haces 6 años, lo que sería comparable en precio a hoy en día al procesador del equipo 1 (un i5 de octava generación) pero sacando mucho menos rendimiento en todas las pruebas con notable diferencia.

Bibliografía

- stackoverflow.com. How to use QueryPerformanceCounter?: disponible aqui.
- wikipedia.org. Anexo:Instrucciones x86: disponible aqui.
- exabyteinformatica.com. Miquel Albert Orenga, Gerard Enrique Manonellas. Programación en ensamblador (x86-64). Universitat Oberta de Catalunya: disponible aqui.
- Universidad Nacional de Córdoba (UNC) - Facultad de Matemática, Astronomía y Física, Argentina. Operaciones SIMD: disponible aqui .
- docs.oracle.com. SSE Instructions: disponible aqui.
- spec.org. SPEC y SPEC cpu2000: disponible aqui.