# FASE II

Benchmark reducido y benchmark sintético

**Arquitectura de los Computadores**

**Lunes 17:00 a 19:00**

**Controlador: Alberto Sapiña Mora**

**Director: Jorge Núñez González**

**Auxiliar: Pablo Requena González**

**Secretario: Marcos González Verdú**

**ÍNDICE**

1. Introducción………………………………………………………………pág. 3
2. Benchmark Reducido…………………………………………………pág. 4
3. Benchmark Sintético………………………………………………….pág. 7
4. Evaluación de los resultados ……………………………………. pág. 7
5. **Introducción**

* ¿Qué es un benchmark?

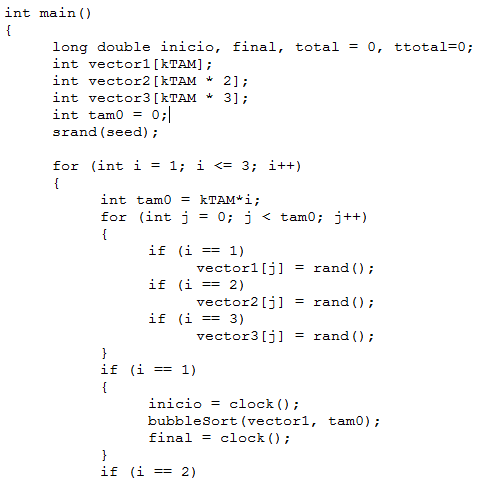
Es un conjunto de técnicas específicas que nos permite evaluar el rendimiento de un sistema computacional. Varían dependiendo de lo que queramos evaluar, por ejemplo, no se emplearán las mismas técnicas para evaluar un ordenador, un componente de un ordenador, una aplicación o una serie de instrucciones.

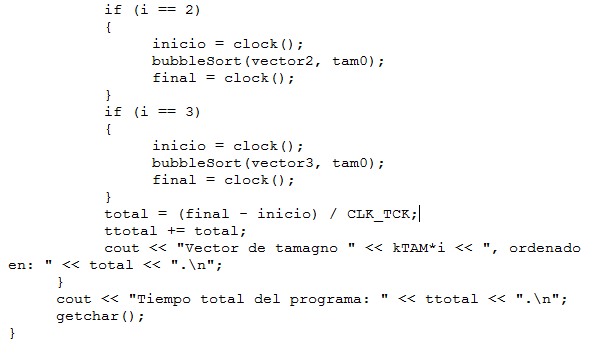
* ¿Qué tipos de benchmarks existen?
  + Reducido: destinados simplemente a medir componentes básicos de un computador (latencia de memoria, megahercios, etc.)
  + Sintéticos: evalúan una capacidad concreta de un subsistema, es decir, a nivel componente. Está compuesto por un conjunto de operaciones básicas, ejecutadas para simular posibles situaciones del subsistema para medir el rendimiento de este. Por ejemplo, puede contener varias operaciones read/write en un disco duro para medir el rendimiento de este subsistema.

1. **Benchmark reducido**

Como benchmark reducido hemos utilizado el algoritmo burbuja, con tres vectores de distintos tamaños, y hemos calculado el tamaño que tarda en ordenar el vector desordenado, rellenado de números aleatorios. Hemos elegido este algoritmo como Benchmark reducido para comprobar la velocidad de carga e intercambio de elementos en la memoria, que ocurren gran cantidad de veces durante la ejecución del algoritmo.

En primer lugar analizamos el main:





Lo primero que hacemos es crear 3 vectores en base a una constante kTAM que en este caso es 5000, es decir, creamos tres vectores de tamaño 5000, 10000 y 15000.

Después el for se ejecutará tres veces (una por cada vector) y dentro rellenamos de números aleatorios el vector correspondiente y llamamos al algoritmo de ordenación.

Antes y después almacenamos el valor de clock() en inicio y final, correspondientemente, para poder obtener el tiempo que ha tardado el algoritmo haciendo (final – inicio)/CLK\_TCK.

Después de esto, sumamos a total el valor obtenido para poder devolver el tiempo total de ejecución del Benchmark.

El algoritmo BubbleSort:

int aux=0;

for(int i=0; i<tam; i++)

for(int j=tam-1; j>i; j--)

if(vector[j]>vector[j-1]

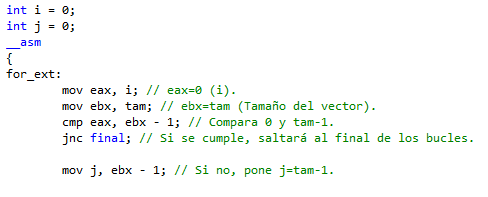
{

aux=vector[j];

vector[j]= vector[j-1];

vector[j-1]=aux;

}



- Inicializamos i y j a 0 para trabajar con ellos, más tarde.

- Creamos el bubble exterior, al cual llamaremos for\_ext.

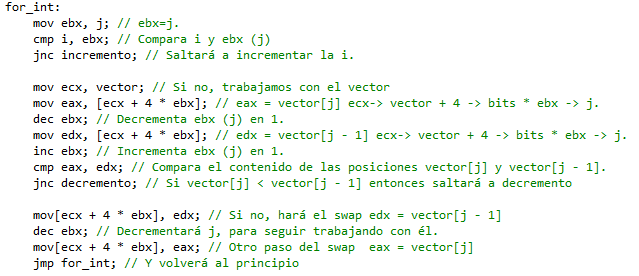
- Movemos a eax el contenido de i.

- Movemos a ebx el contenido del tamaño del vector (que se le ha pasado por referencia).

- Comparamos eax (i) con exb-1 (que es el tamaño del vector).

- Si se cumple la condición terminará el algoritmo.

- Si no, continuará. Moviendo a j el contenido de ebx – 1 (tam – 1).



- Creamos el segundo bucle, lo llamaremos for\_int.

- Moveremos a ebx el contenido de j (tam – 1).

- Compararemos i con ebx.

- Si se cumple, incrementará la i (en incremento, el cual se explicará después).

- Si no, seguiremos, esta vez, trabajando con el vector-

- Moveremos a ecx el vector con el que trabajamos.

- Moveremos a eax la primera posición que queremos comparar, como es ecx + 4\*ebx, pues estamos en el vector[j] y 4 es el número de bits que ocupa un entero.

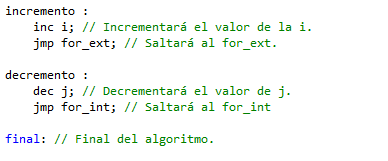
- Tras ello decrementamos j para comprar con la posición anterior.

- Y volvemos a guardar la posición, esta vez en edx.

- Tras esto, incrementamos j (ebx) y compararemos el contenido de las posiciones vector[j] y vector[j – 1].

- Si se cumple, saltaremos a decrementar la j, por lo cual miraremos las siguientes posiciones.

- Si no, haremos el swap, saltaremos al principio del buble interior, y volveremos a empezar.



Estos dos bucles son los que controlan la posición de “i” y de “j” para que se realizan los bucles.

1. **Benchmark sintético**

Como benchmark sintético hemos simulado un conjunto de operaciones aritméticas en coma flotante y hemos sacado el tiempo medio que tarda el computador en ejecutar todas las operaciones. Para ello hemos utilizado 3 números en coma flotante (a, b y c), y hemos realizado varias sumas, restas, multiplicaciones, divisiones (y también operaciones de raíz y potencia basadas en las de multiplicación y división).

En cuanto al código auxiliar hemos hecho un bucle en el que llamamos x veces al bloque de código en ensamblador anteriormente descrito y calculamos la media del tiempo de ejecución de este.

Hemos orientado este benchmark a testear computadores para programas con fines científicos/matemáticos. Por eso, tiene un 78.57% de instrucciones aritméticas y un 21.43% de control (he considerado despreciables las instrucciones fuera del bucle para los porcentajes ya que se ejecuta un número muy elevado de veces)

1. **Evaluación de los resultados**

Intel Core i5 2450M (2x2.50GHz): 3.273833 segundos

Intel Pentium g840 (2x2.8GHz): 3.574333 segundos

Intel Core i5 M460 (2x2.53 GHz): 4.006500 segundos

Intel Core 2 Duo E6600 (2x 2.40Ghz): 7.039167 segundos