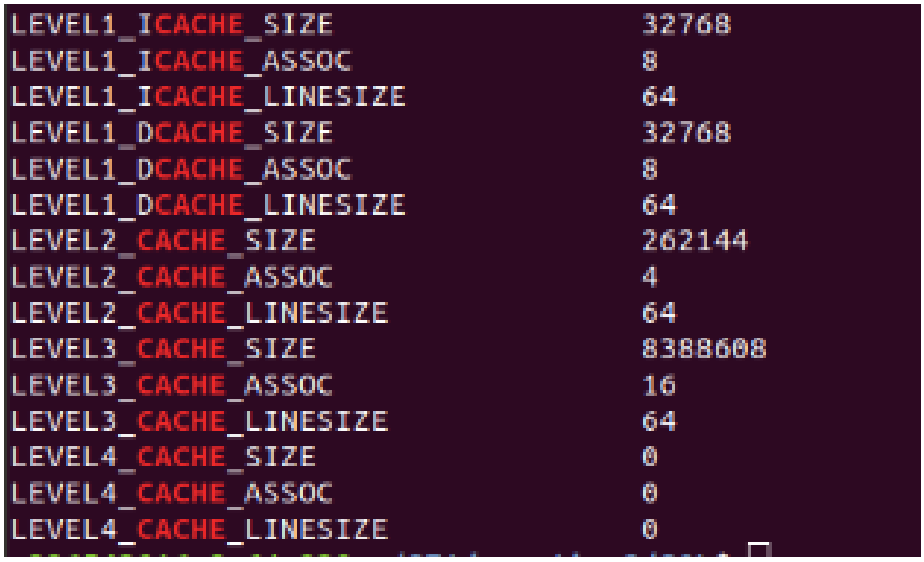
ARQUITECTURA DE ORDENADORES

Práctica 3

Memoria Caché y Rendimiento

Jorge Santisteban Rivas
Javier Martínez Rubio
Curso 2018-2019

Ejercicio 0 :Información sobre la caché del sistema



Existen diferentes niveles para guardar los datos de la memoria caché, alojandose estos en uno o en otro dependiendo de la frecuencia de uso que tenga. En nuestro caso, podemos observar que existen 4 niveles distintos, aunque el último no se utiliza para nada.

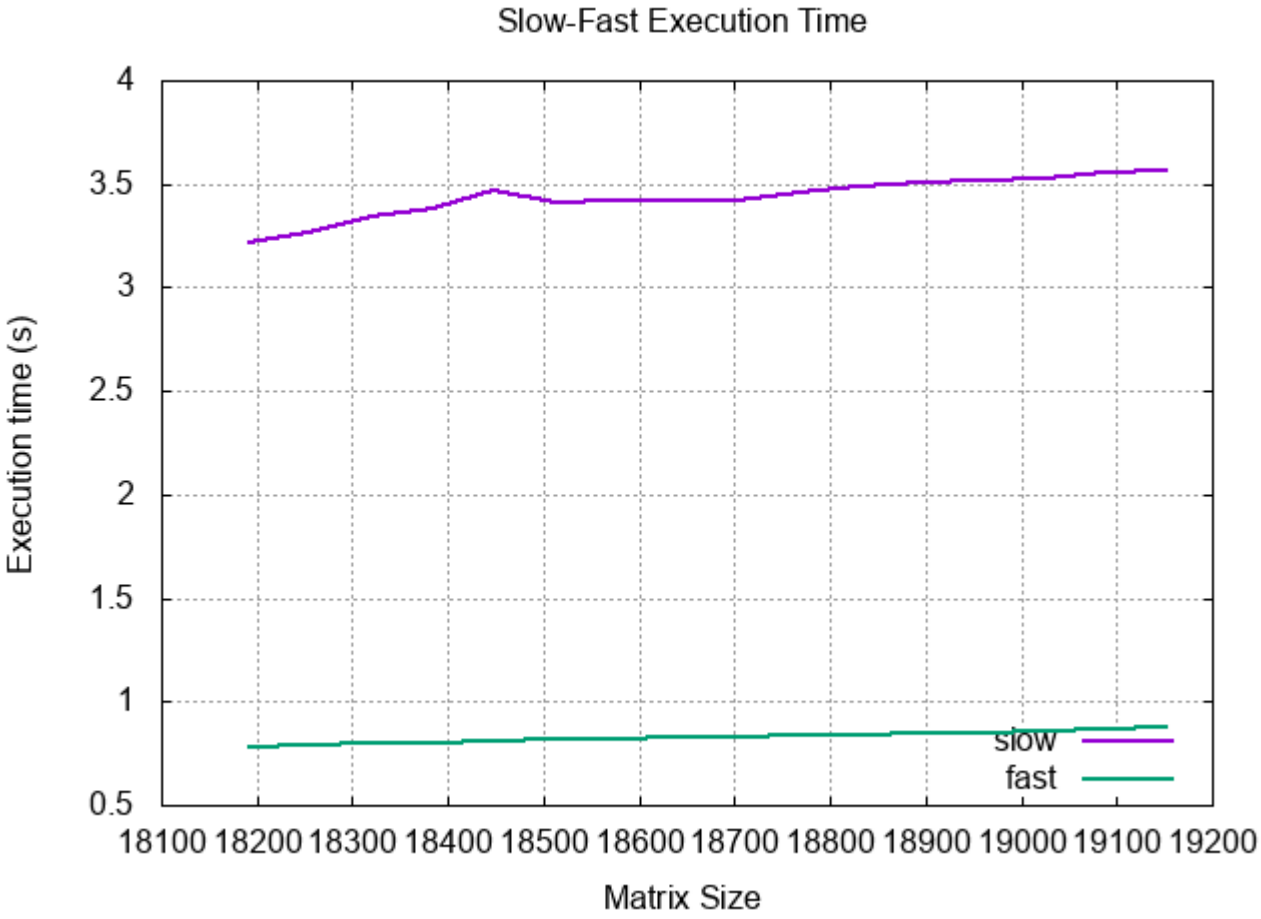
El primer nivel es el único que está subdividido en dos niveles, uno para almacenar datos y otro para almacenar instrucciones. Este nivel es el que ofrece un tiempo de respuesta más rápido, ya que su contenido es el de uso más frecuente. Así, podemos observar que para ambos casos, el tamaño de caché es de 32768 bytes, asociativa de 8 líneas y tamaño de línea de 64 bits.

Los dos siguientes niveles de memoria caché no están subdivididos, como si lo está el primero. Estos almacenan datos menos utilizados, y, a medida que aumentan los niveles, disminuye la rapidez. Observamos que el tamaño de línea se preserva en los tres niveles, siendo este 64 bits.
El tamaño de vías cambia, siendo 4 para el nivel 2 y 8 para el nivel 3. A su vez, observamos también un gran incremento en el tamaño de caché, siendo 262144 bytes para el segundo nivel y 8388608.

Ejercicio 1: Memoria caché y rendimiento

En primer lugar, comentar que la toma de medidas la realizamos múltiples veces para tomar los datos con una mayor precisión. Esto se debe a que si solamente lo ejecutamos una vez, el resultado no va a ser demasiado representativo, ya que la velocidad con la que el ordenador ejecuta el programa depende de su estado en ese momento, y de los demás procesos que tenga que intercalar. Así, una toma de medidas más exacta es ejecutarlo más veces y tomar la media de los resultados. En nuestro caso, el número de repeticiones es 14.

A continuación, mostramos en un gráfico los valores obtenidos:



Observamos cómo claramente el programa fast es mucho más rápido que el slow. Mientras que el fast varía más o menosentre 0,8 segundos para matrices de menor tamaño y 0,9 segundos para matrices de mayor tamaño, el programa slow tarda 3,2 segundos y 3,6 segundos, aproximadamente, en hacer lo mismo.

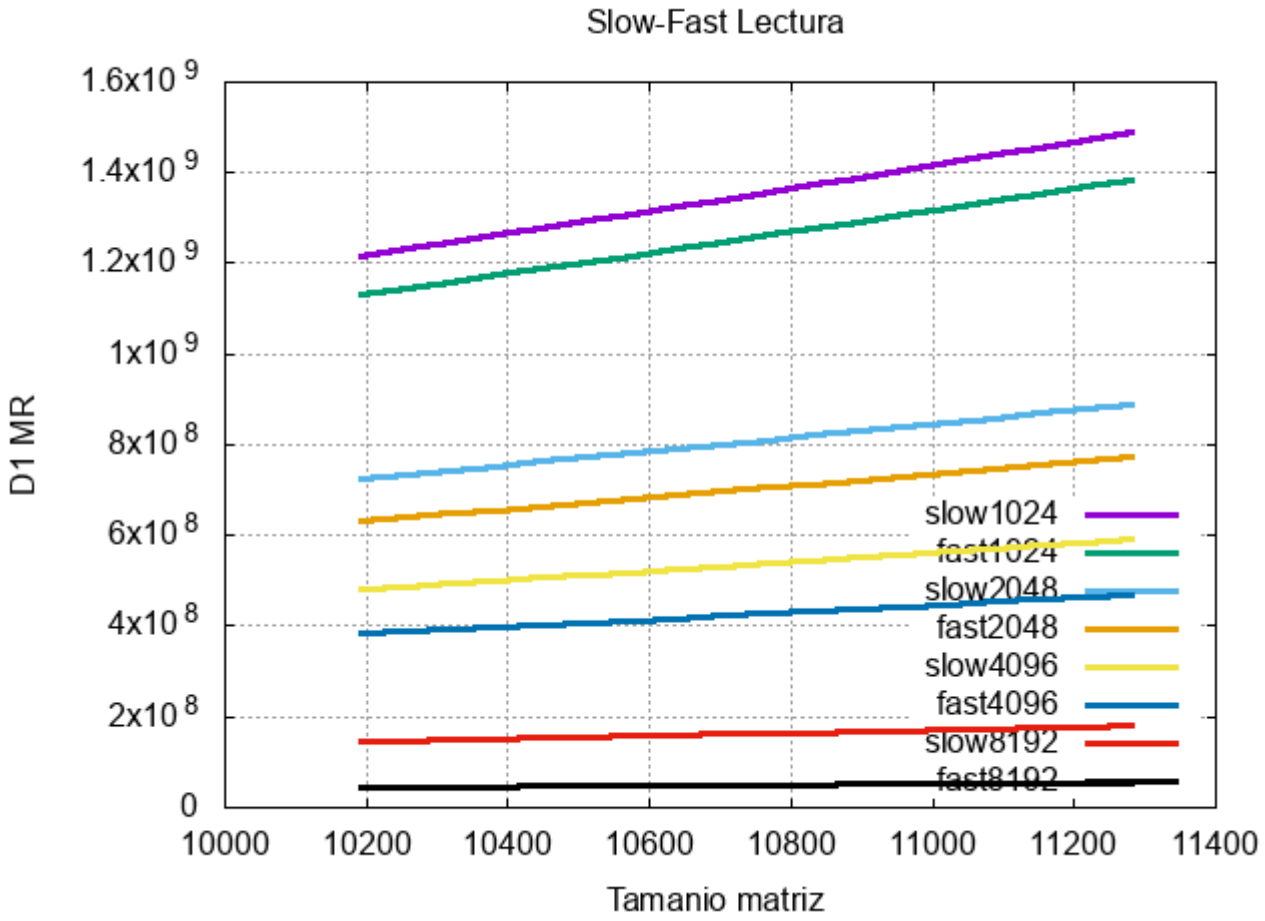
Para obtener estos resultados, hemos intercalado los diferentes tamaños de las matrices, para así evitar que la memoria caché tenga guardado el resultado de la matriz que se está ejecutando, con lo que obtendríamos un tiempo demasiado pequeño, que no se acercaría demasiado al tiempo real.

En la gráfica adjunta, observamos que los tiempos de ejecución de ambos programas empiezan siendo ya muy distintos. Esto se debe a que el primer tamaño de matriz es 18192, un tamaño demasiado grande en el que ya se nota mucho la diferencia entre ambos programas. Para comprobar si con matrices pequeñas los tiempos son más parecidos, ejecutamos ambos programas con un

tamaño de 3000, y obtenemos que el slow tarda 0,06428 segundos y el fast 0,0277. Estos son unos tiempos considerablemente parecidos.

En la caché de datos, los elementos de una matriz se encuentran, ordenandos por filas, en direcciones consecutivas. En el caso en que la matriz tenga una dimensión pequeña, un mismo bloque podrá alocar todos estos elementos, por lo que se accederá de forma inmediata a todos los elementos de la matriz una vez el bloque esté en el nivel 1. Esto nos da un tiempo de ejecución pequeño para ambos programas. Sin embargo, cuando la matriz tiene una mayor dimensión, serán necesarios más bloques para albergar los elementos de esta. En este caso, el programa slow, que realiza la suma por columas, necesitará actualizar la caché constantemente con el bloque en el que se encuentre el dato en ese momento. Además al necesitar los elementos de cada columna, los datos están más separados en la caché, por lo que es necesario consumir más tiempo. Por su parte, el programa fast, al sumar los datos por filas, podrá encontrar dichos datos en un mismo bloque, por lo que no necesita actualizar la caché tantas veces como requiere el programa slow, y podrá acceder a estos datos de una forma mucho más inmediata.

Ejercicio 2:Tamaño de la caché y rendimiento

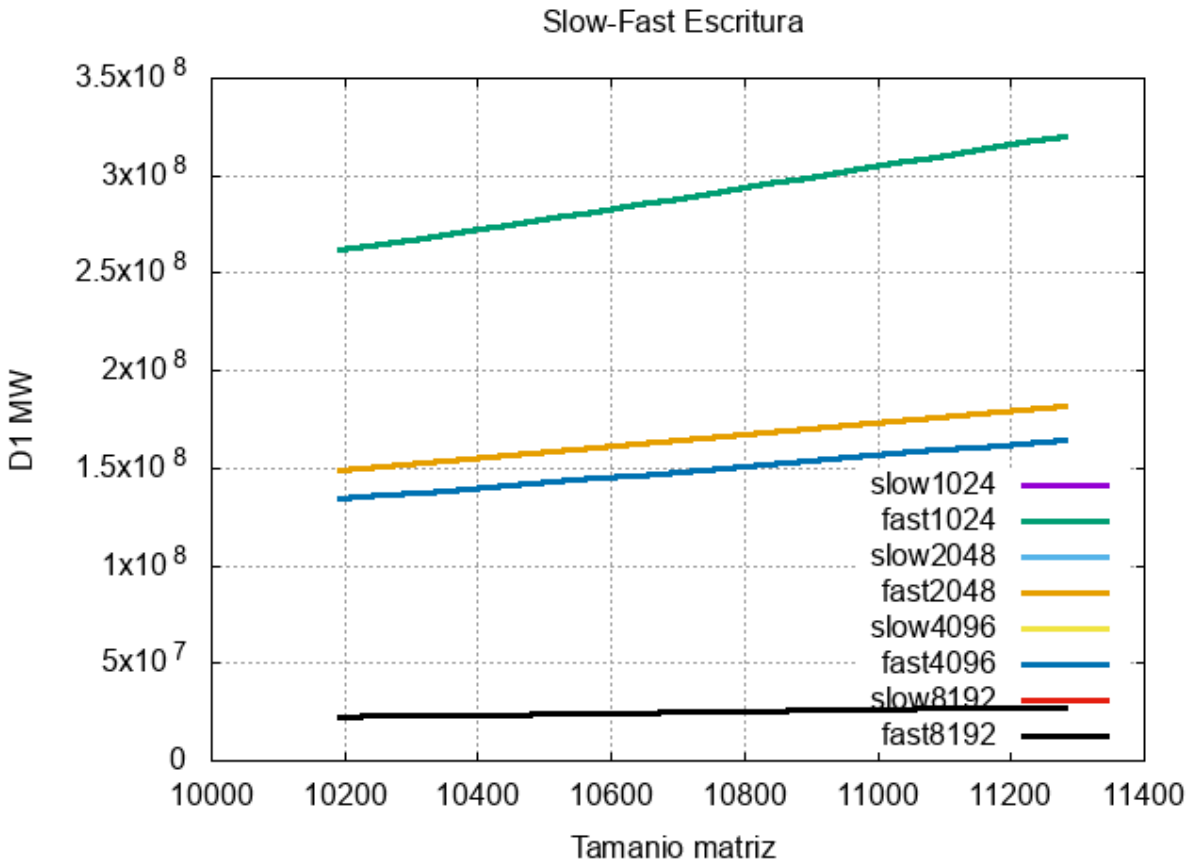


Cuando queremos leer un dato que no se encuentra en la caché, de forma que esta se tenga que actualizar para llevar a cabo la búsqueda de dicho dato, se produce un fallo de lectura.

En esta gráfica, se puede observar claramente que estos fallos aumentan a medida que las matrices a operar son más grandes. Esto ocurre ya que al ser la matriz más grande, harán falta más accesos a memoria, puesto que harán falta más bloques para almacenar los elementos.

A su vez, se ve que el programa fast produce menos fallos que el slow para cualquier tamaño de caché. Esto es bastante lógico ya que el fast, como recorre las matrices por filas, los datos que necesita se encuentran seguidos en un mismo bloque, y no tendrá que leer otro hasta que lo haya leído entero. Por su parte, slow tendrá que cambiar de bloque cada vez que lee un dato de la misma columna, lo que requerirá más actualizaciones, y, por consiguiente, más fallos.

También se aprecia que, al aumentar el tamaño de caché, se reducen los fallos de lectura. Esto se debe a que si la caché tiene más tamaño, también lo tendrán los bloques, lo que aumenta la probabilidad de que el dato que queremos se encuentre en el nivel 1.



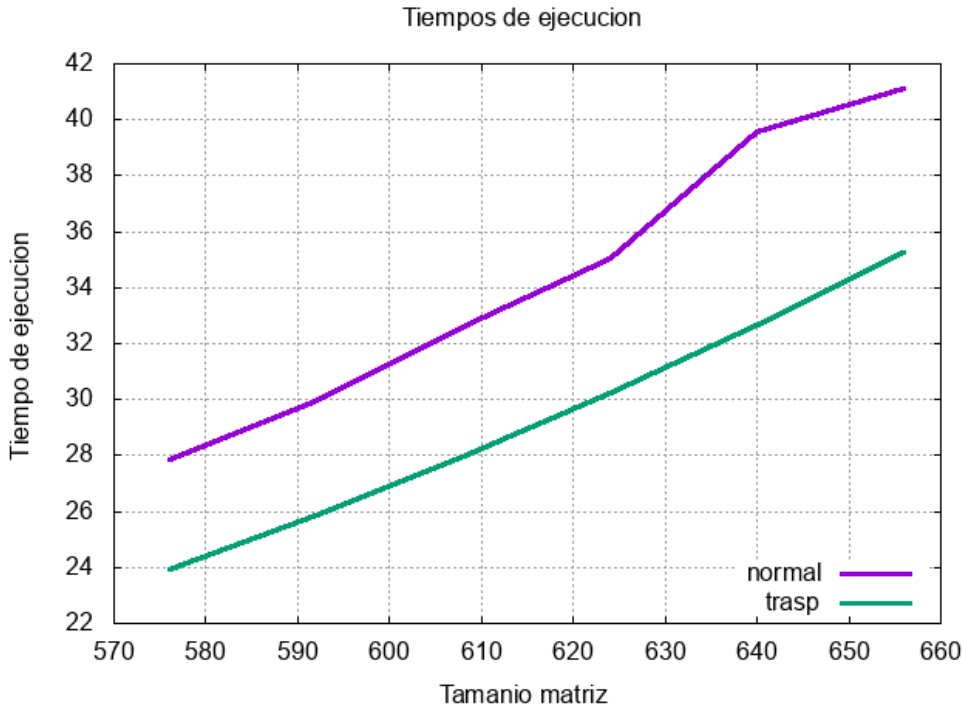
Un fallo de escritura se produce cuando el dato a escribir en caché no tiene hueco en memoria.

Lo más destacable de esta gráfica es que los dos programas producen los mismos fallos de escritura, a diferencia de los fallos de lectura. Esto se debe a que ambos programas solamente escriben una vez por cada elemento de la matriz, lo que provoca el mismo número de fallos para ambos.

Se aprecia también que, al igual que con los fallos de lectura, y por las mismas razones, los fallos de escritura aumentan a medida que aumenta el tamaño de la matriz, y disminuyen cuando aumenta el tamaño de la caché.

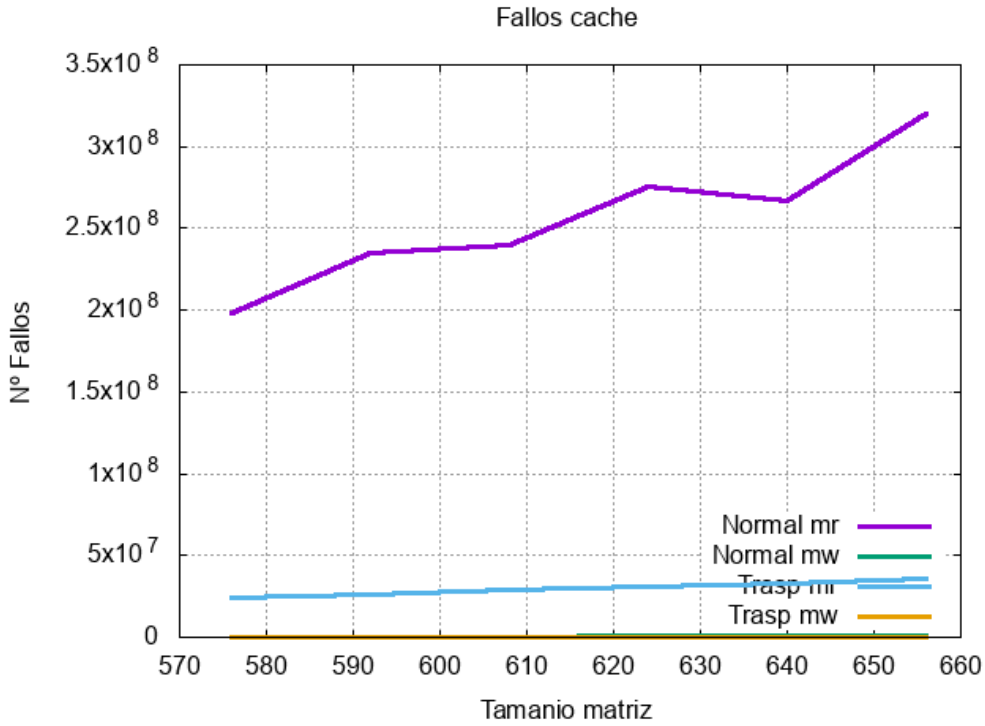
Si comparamos la cantidad de fallos de lectura y de escritura, vemos que hay considerablemente menos fallos de escritura. Esto se debe a que la acción de escribir solamente se realiza una vez al crearla, mientras que son necesarias muchas más lecturas, dependiendo esto también del algoritmo utilizado.

Ejercicio 3: Caché y multiplicación de matrices



Observamos cómo aumenta el tiempo de ejecución al aumentar el tamaño de la matriz, ya que aumentan el número de operaciones que se realizan.

También vemos que el rendimiento para las matrices traspuestas es mejor, ya que, a pesar de tener que trasponer la matriz, multiplicar fila a fila es mucho más rápido que hacerlo fila por columna, lo que necesita muchas más actualizaciones de caché. Por tanto, la multiplicación de matrices traspuestas es más rápida.



En la gráfica podemos ver que el número de fallos de lectura en la multiplicación de matrices normales es mayor, al igual que su crecimiento con respecto a la dimensión de la matriz. Como en el caso anterior, esto se debe a que la multiplicación fila por fila es mucho más rápida.

Por otro lado, la cantidad de fallos de escritura es muy baja. Sin embargo, si miramos el fichero detenidamente, podemos ver que el número de fallos de escritura de la multiplicación normal de matrices es menor que el de la multiplicación de manera traspuesta. Esto es porque en la traspuesta se guarda la matriz dos veces, siendo la primera la original y la segunda la traspuesta de esta.

.

Por otro lado vemos que existe una gran diferencia entre el número de fallos lectura con respecto al de escritura, ya que, como hemos comentado con anterioridad, la escritura la realizamos una sola vez en la creación de la matriz mientras que cada vez que se utiliza un elemento de la matriz se ha de leer y puede ocasionar un fallo.