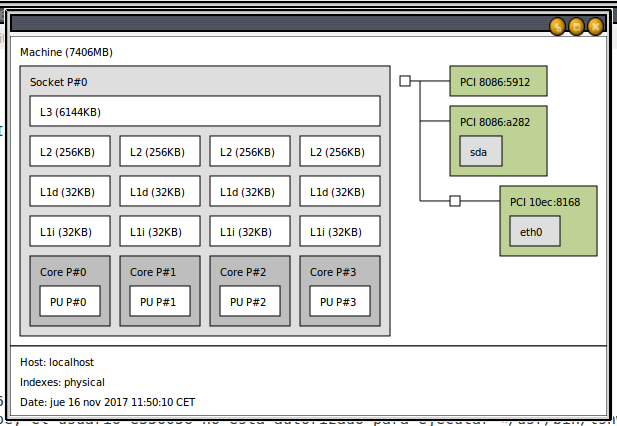
|  |
| --- |
| Arquitectura de Ordenadores |
| Práctica 3: Memoria caché y rendimiento |
| UAM 2017-2018 |

|  |
| --- |
| Óscar Gómez Borzdynski  José Ignacio Gómez García |

Ejercicio 0:

Para este ejercicio necesitamos conocer la estructura de caché de nuestro sistema. Para ello utilizamos el comando *lstopo*, que proporciona una representación gráfica de la caché del sistema. Para obtener más información, como la asociatividad de cada nivel, utilizaremos *getconf -a* y buscaremos el término cache.



Vemos que nuestra máquina tiene 4 núcleos con una caché a 3 niveles:

**Nivel 1:** Tenemos una cache de 256 KB asociativa de 8 vías. Está dividida en 8 bloques de 32 KB, 2 por núcleo, con separación entre datos e instrucciones. Dados estos datos podemos suponer que el procesador tiene una arquitectura tipo Harvard.

**Nivel 2:** Tenemos una cache de 1 MB asociativa de 4 vías. Está dividida en 4 bloques de 256 KB, uno por núcleo, sin separación entre datos e instrucciones.

**Nivel 3:** Tenemos una cache de 6MB asociativa de 12 vías. En este caso no se encuentra dividida en bloques por núcleo, sino que es compartida entre todos ellos.

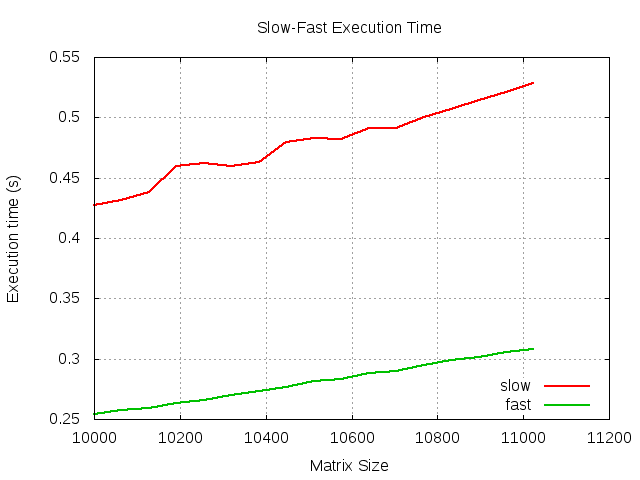
Ejercicio 1:

Nótese que a partir de este ejercicio utilizaremos P=0, en vez de P=6, por indicación del profesor.

Para este apartado utilizaremos un script que realizará un número de iteraciones determinados. Para definir este dato, se deberá modificar la variable Iterations. En nuestro caso haremos 10 iteraciones.

Además de ello utilizaremos matrices de 10000 a 11024 en saltos de 64. Escribiremos los tiempos de ejecución de los programas slow y fast en un fichero temporal. Obtendremos sus medias y luego los ordenaremos para proceder a realizar la gráfica solicitada.

La manera de obtener los datos de la manera más veraz es intercalar los programas. Para ello, ejecutaremos siempre un slow seguido de un fast para cada tamaño de matriz.



En la gráfica se puede apreciar una diferencia notable en el tiempo de ejecución entre slow y fast. Debido a que el programa hace básicamente lo mismo, creemos que esta diferencia se debe a los fallos de caché. La diferencia en el orden de acceso a los datos es lo que provocará esta diferencia.

Entrando más en detalle podemos observar que la diferencia de tiempo para matrices de 10000 es 0.173s, mientras que para matrices de 11024 de 0.219s. Pensamos que según aumentamos los tamaños de las matrices, el número de datos accedidos aumenta. Por ello, debido a que el programa slow tiene una forma de almacenamiento menos eficiente, la diferencia en los fallos de caché entre ambas aumenta.

Para comprobar por qué sucede esto, procedemos a analizar el código proporcionado. Observamos que la rutina que crea la matriz mediante un array de dimensión “n x n”. Posteriormente reasigna las direcciones a un array de n punteros, dividiendo dicho array en segmentos de n elementos. Por ello, vemos que los datos se almacenan por filas. Siendo todos los datos de una fila contiguos en memoria.

Vemos que el programa fast suma los elementos pro filas, siendo estos contiguos en memoria. Por ello, al cargar el primer dato, se carga el bloque entero, disponiendo del resto de datos del bloque en la cache y evitando fallos posteriores. En el caso del programa slow, accede por columnas a los datos, provocando fallos en casi todos los accesos a memoria.