Jerarquía de memoria y comportamiento de memoria caché

Carlos Cao López, Pedro Vidal Villalba

Arquitectura de Computadores
Grupo 01
{carlos.cao,pedro.vidal.villalba}@rai.usc.es

11 de marzo de 2024

I. Descripción del experimento

Cada uno de los experimentos ha sido realizado en el supercomputador FinisTerrae III del Cesga. En el momento de la ejecución de los experimentos este cuenta con las características técnicas expuestas en la tabla 1.

Para cada ejecución del programa `acp1.c` se ha solicitado un nodo con 64 cores (virtuales, 32 cores físicas que son las que tiene cada procesador) y 256 GiB de memoria. De cada ejecución hemos obtenido el número medio de ciclos por acceso a memoria y este ha sido representado frente al número de lineas caché accedidas (en escala logarítmica) en las figuras 1, 2 y 3.

Para el análisis de los efectos de la localidad y la precarga, se han tomado distintos valores de D potencias enteras de 2 y de L en función del tamaño de las cachés L1 y L2. En las gráficas se representan la media y desviación típicas de los ciclos por acceso para cada valor de D y L.

II. Resultados

Se presentan a continuación los resultados obtenidos tras las ejecuciones de los experimentos usando tipo de datos `double` y con acceso indirecto a través de un array de índices en la figura 1; usando tipo de datos `int` y acceso indirecto en la figura 2; y tipo de datos `double` y acceso directo en la figura 3.

Los valores de D seleccionados, que se pueden ver en las gráficas, fueron elegidos para observar la diferencia en la localidad de acceso a los datos. Así, para valores pequeños ($D = 1$ y $D = 8$), se realizan varios accesos consecutivos al array `A` dentro de la misma línea caché o en líneas caché consecutivas, con lo que podemos observar claramente un rendimiento máximo al aprovechar la localidad espacial de los datos. A partir de ahí, se escogieron valores de D más grandes en los que los accesos consecutivos a `A` se saltasen líneas caché, dificultando la labor del *prefetching* y afectando directamente a la localidad espacial de los datos así como a la temporal, pues conllevan un array más grande que no se pueda almacenar entero en la caché y, por tanto, que tengan que producirse remplazamientos de líneas caché, lo que se traduce en mayores tiempos de ejecución.

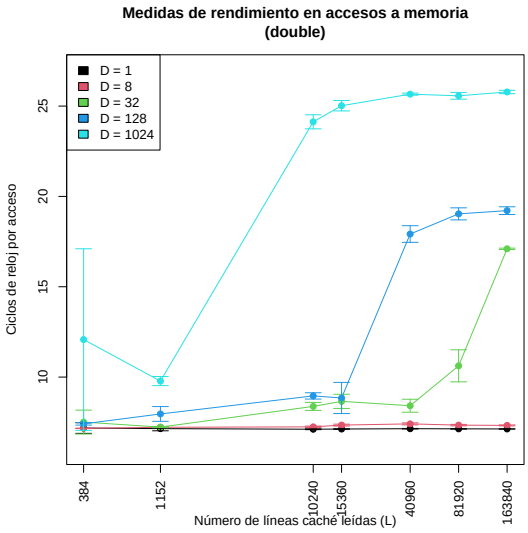

Figura 1: *Resultados con acceso indirecto y* `double`

A esto último también afecta significativamente el valor de L, pues a medida que este aumenta, lo hace al tamaño del array `A`, con lo que no cabe entero en memoria y no se puede aprovechar la localidad temporal de los datos

Procesador	Intel Xeon Ice Lake 8352Y de 32 cores, 2.2GHz
Tamaño de línea caché	64 bytes
Tamaño de la caché L1	49 152 bytes
Tamaño de la caché L2	1 310 720 bytes
Compilador	gcc, *(Gentoo 10.1.0-r2 p3) 10.1.0*
Shell	GNU bash, *version 5.0.18(1)-release (x86_64-pc-linux-gnu)*
Sistema operativo	Rocky Linux, *release 8.4 (Green Obsidian)*
Kernel de Linux	4.18.0-305.3.1.el8_4.x86_64

Cuadro 1: *Características FinisTerrae III*

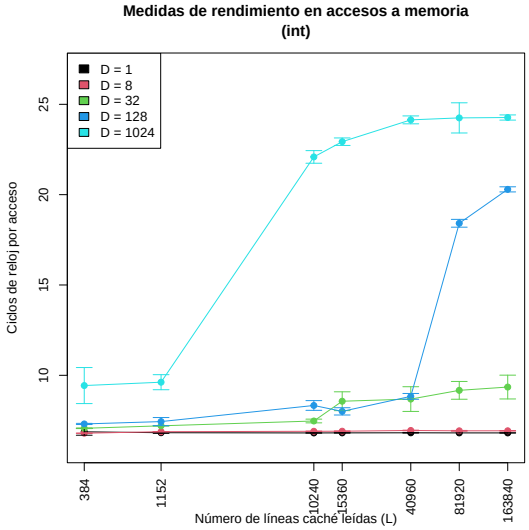

Figura 2: *Resultados con acceso indirecto e* `int`

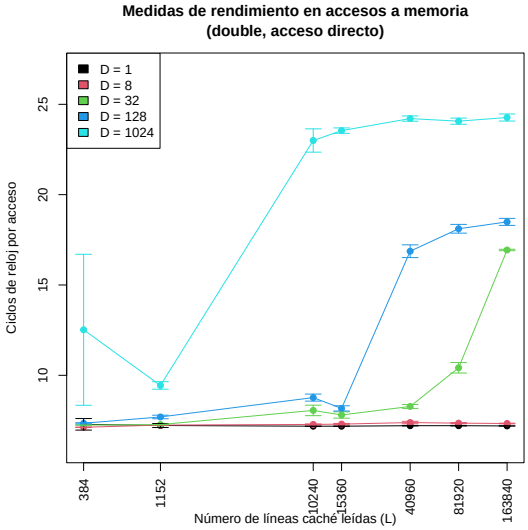

Figura 3: *Resultados con acceso directo y* `double`

entre las distintas vueltas del bucle. Así, deben producirse remplazamientos en la caché, con su correspondiente penalización temporal. Esto explica la tendencia ascendente en el número de ciclos de reloj que se observa en todas las gráficas en función del valor de L.

Observamos también que cuánto mayor es el valor de D y, por tanto, más espaciados se encuentran los datos a los que accedemos en memoria, antes ocurren ciertos fenómenos comunes a las tres gráficas como el comportamiento asintótico hacia el final de las mismas o la subida brusca de los ciclos medios. Esto es fácilmente explicable pues, cuanto mayor es el valor de D, más espaciados están los datos que se leen y más atenuados se ven los efectos del *prefetching* así como de la localidad espacial de los datos.

La precarga (*prefetching*) juega también un papel importante en los resultados observados. La precarga es un mecanismo por el cual se cargan en los niveles más altos de la jerarquía de memoria datos que todavía no han sido solicitados, con la esperanza de que se necesiten más adelante y tenerlos ya disponibles. Se basa, como el funcionamiento mismo de las cachés, en el principio de localidad.

La precarga a la LLC (*last level cache*, caché L3) desde memoria principal aumenta el rendimiento general del programa. La precarga a la caché L2, por la cual se cargan dos líneas caché consecutivas en lugar de una, explicaría parte de la velocidad de los accesos para los valores más pequeños de D, pues para estos los accesos sucesivos están en líneas caché contiguas. La precarga en la caché L1 predice si se realizan múltiples accesos a posiciones de memoria equiespaciadas (como estamos haciendo en el programa), y trae a memoria las líneas caché siguientes, siempre y cuando

estén dentro de la misma página de memoria principal (4 KiB); esto explica la diferencia de velocidad para el valor más grande de D, en el que este mecanismo no puede actuar por estar cada dato consecutivo en una página diferente.

La subida más significativa en las gráficas de los `double` para $D = 1024$, como se puede ver en la figura 1, sucede cuando se llena la caché L2. Como se ha mencionado antes, debemos tener en cuenta que la caché L2 hace *prefetching* de dos lineas caché de cada vez. Si a esto le sumamos el hecho de que los datos, debido al alto valor de D, están muy separados y, por tanto, nunca se traen lineas caché duplicadas, tenemos que accediendo a 10240 líneas distintas, la caché L2 se habrá llenado lo suficiente para dejar notar los efectos adversos pues, debemos tener en mente que estamos tratando con una caché asociativa por conjuntos y, por tanto, la mayoría de estos conjuntos se habrá llenado antes de que lo haya hecho el total de la memoria caché.

Con los `int`, debido a que ocupan la mitad que los `double` y como se puede apreciar en la figura 2, ocurre lo mismo para $D = 1024$. Sin embargo, para valores de D menores como $D = 128$ o $D = 32$, observamos que se junta la mayor carga de datos por línea con el menor espaciamiento de los mismos postergando así los efectos en el tiempo.

El comportamiento asintótico se produce una vez el número de lineas caché que se acceden supera al número de lineas caché de la memoria L2 para los valores $D = 128$ y $D = 1024$, lo cual resulta consistente pues, de ese punto en adelante, la localidad temporal se ve bruscamente reducida por la necesidad de sustituír lineas caché con otros datos.

Se observa también una importante variabilidad en los datos para el valor más grande de D, en el que los tiempos son mayores, y el más pequeño de L. Esto es esperable, pues se realizan pocos accesos y pequeñas variaciones aleatorias no tienen la oportunidad de diluirse entre los datos.

Finalmente, es importante mencionar que no se observan diferencias significativas entre el uso de `double` o `int` ni entre el uso de acceso indirecto a través de un vector de índices o el acceso directo en lo que se refiere a los patrones de comportamiento de los tiempos de ejecución. Sí se ven, como es lógico, ligeras mejoras de rendimiento globales en el uso de `int`, pues las operaciones con este tipo de datos son algo más rápidas que las que se efectúan con `double`.

Se observa también unos tiempos ligeramente menores de ejecución en general cuando se usa acceso directo que cuando se hace acceso indirecto. Esto es fácil de explicar, pues con el acceso directo cada acceso al array `A` solo necesita hacer una multiplicación de dos enteros, que es más rápida que un acceso a memoria al array de índices, aunque el dato esté previamente calculado. Sin embargo, la forma general de las gráficas es exactamente igual en ambos casos.

III. Conclusiones

Se realizaron diferentes experimentos para el estudio de la jerarquía de memoria, las memorias caché y el efecto de la localidad en el acceso a los datos. Sistemáticamente se observa que la localidad de los datos, tanto espacial como temporal, juega un papel clave en el rendimiento de los programas, aun cuando se realizan operaciones tan simples como la lectura de datos equiespaciados en un array.

Así, tanto el incremento en la distancia entre los datos leídos como el incremento del número de datos que se leen conllevan una menor localidad espacial y un llenado temprano de las memorias caché de los niveles superiores, resultando en notables penalizaciones en el rendimiento.

Acceder a datos en posiciones lejanas de memoria también afecta de forma negativa a la capacidad del procesador de precarga datos, resultando en reducciones notables de rendimiento.

Por último, el uso de tipos de datos enteros supone también una ligera ventaja sobre las variables en punto flotante de doble precisión, pues el ordenador puede manejarlos con mayor velocidad. También se observa un aumento de eficiencia al acceder directamente a los elementos del array en lugar de hacerlo de forma indirecta mediante un array de índices.