Estudio del efecto de la localidad en accesos a memoria

Pablo Mouriño Lorenzo, Carlos Moldes Peña

Arquitectura de Computadores
Grupo 03
{pablo.mourino.lorenzo,carlos.moldes}@rai.usc.es

Resumen

El estudio que ha sido realizado analiza la variación del efecto de localidad en el acceso a memoria. Para esto, probamos distintos valores de desplazamiento por las líneas de la caché para observar bien este suceso. Además del desplazamiento, también tenemos en cuenta un mínimo de líneas caché diferentes a las que queremos acceder, variando así la alineación de los datos.

Palabras clave: caché, localidad, ciclos, accesos, doubles, enteros,...

I. Metodología

Para el siguiente trabajo, quisimos analizar cómo puede variar el tiempo por acceso a los datos que se encuentran en las líneas caché teniendo en cuenta el efecto de la localidad a partir de un programa en C, donde realizamos la suma de R elementos de un vector, de tamaño R*D, el cual almacena `doubles` (A[]) que están alineados dependiendo del número de líneas (L) y a los cuales accedemos con un desplazamiendo (D). Esta suma se repite un total de diez veces para poder sacar el valor medio de ciclos y el número de ciclos medio por acceso a memoria de las distintas medidas. Tras esto, nuestro objetivo fue sacar una conclusión razonable al ir variando los parámetros indicados.

Para realizar todo esto, elegimos los siguientes valores: en el caso de D, deben ser cinco valores que sean potencia de 2. Nosotros escogimos los siguientes: 1 (obtenemos la máxima localidad), 4 (Realiza dos lecturas en cada línea, manteniendo aún cierta localidad), 32 (en este momento se saltan varias líneas enteras con una lectura, perdiendo bastante localidad), 64(mismo caso que antes, pero cada vez se nota más la pérdida de localidad) y 128 (mismo caso que los dos anteriores, pero se pierde mucha localidad). Para R cogemos 7 medidas distintas de cada valor de D. Para ello, antes calculamos los valores de L siempre de la misma forma: {0.5*S1, 1.5*S1, 0.5*S2, 0.75*S2, 2*S2, 4*S2, 8*S2} , siendo S1 el número de líneas caché que caben en la caché L1 de datos y S2 el número de líneas caché que caben en la caché L2. Para la obtención de estos valores, tenemos varias posibilidades; bien sea acceder a la información proporcionada por el fabricante de la CPU o, como es nuestro caso, emplear ciertos comandos por terminal que nos proporcionan esta información. Concretamente, se empleó en este experimento el comado `lscpu` con las flags `-BC`, que nos proporcionan la información solamente acerca de las cachés y en bytes. Es necesario también conocer el tamaño de la línea caché, lo cual se puede obtener con el comando `getconf LEVEL1_DCACHE_LINESIZE`. Los valores de estos parámetros son 64 bytes para el tamaño de línea, 768 líneas en la caché L1, 20480 en la de nivel dos, L2 y las L líneas {384, 1152, 10240, 15360, 40960, 81920, 163840}.

Una vez tenemos los valores de S1, S2, D y el tamaño de línea, debemos obtener los R elementos del vector que nos aseguren el acceso a L líneas de caché diferentes. En el caso de usar `doubles`, en 64 bytes podemos almacenar hasta 8, entonces si elegimos un desplazamiento de D = 1, accederemos a una línea diferente cada 8 accesos, por lo que R = 8L. Este razonamiento lo podemos generalizar con la fórmula $R = (8/D) * L$. Es importante destacar que, si D >8, limitaremos a R =

L, ya que en caso contrario obtendríamos un R <L y esto no nos permitiría el acceso a las L líneas caché diferentes. En caso de usar `ints`, debemos sustituir los 8's por 16's, ya que en 64 bytes podemos almacenar 16 `ints`, ya que cada uno ocupa 4 bytes.

Una vez tenemos todo esto, ya podemos realizar la experimentación en el supercomputador FinisTerrae III. En el siguiente punto se analizarán las gráficas resultado que contienen los resultados de este experimento y de los otros experimentos adicionales que se nos proponen. Por último, se proporcionará una conclusión comentando lo que se ha ido haciendo a lo largo de esta práctica y los resultados a los que hemos llegado.

II. Resultados

En este apartado se tratan los resultados obtenidos en cada uno de los experimentos. El primero de ellos, como se comentó anteriormente, debe ser un vector de `doubles` con la condición de que se acceda indirectamente a través de un vector de enteros *ind[i]* = 0, D, 2D, 3D, etc. Las referencias serán del tipo *A[ind[i]]*. Posteriormente, debemos repetir el mismo experimento pero haciendo el acceso directamente, sin usar el vector *ind[]*. El último de ellos consta de cambiar el vector de doubles por un vector de enteros haciendo nuevamente el acceso de manera indirecta. Una vez realizado esto, obtenemos las tablas que aparecen a continuación donde se observa la comparación de los tres experimentos en cada una de ellas con diferentes colores.

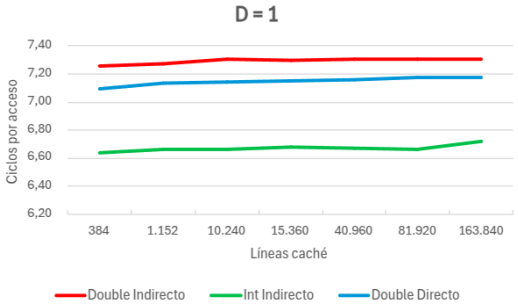

Figura 1: *Gráfica de resultados con D=1*

Tanto en la gráfica 1 como en la 2, obtenemos resultados similares. Se observa claramente que a medida que aumentamos el

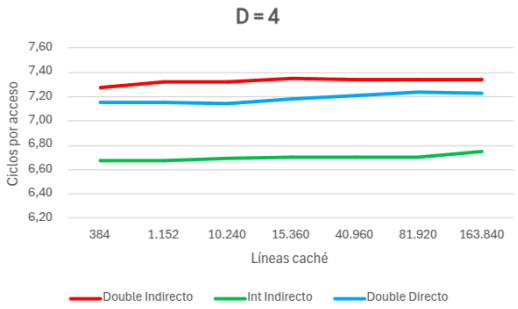

Figura 2: *Gráfica de resultados con D=4*

número de líneas caché que queremos leer, los ciclos por acceso tienen una diferencia mínima en los tres casos, con lo que nos dá a entender de que se mantiene una buena localidad en el caso de D=1, pues es la máxima, y en el caso de D=4 no se aprecia una pérdida importante de localidad, debido a que se realizan varias lecturas de la misma línea, con lo que los resultados son bastante similares.

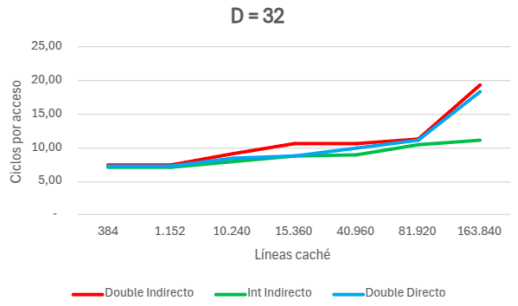

Figura 3: *Gráfica de resultados con D=32*

Ahora, como podemos ver en la gráfica 3, una vez usamos un valor mayor, en este caso D = 32, se pierde ya cierta localidad por lo que, en valores de L muy grandes como el último de todos, es dónde podemos apreciar ya el aumento de ciclos en cada acceso. Es destacable que lo sufren más el tipo de datos `double` que los `int`.

Por último, las gráficas 4 y 5 tendrán un comportamiento similar al caso de D = 32, pero de una forma que el aumento de los ciclos será más prematuro, dado que la localidad que se pierde es aún mayor a medida que D crece.

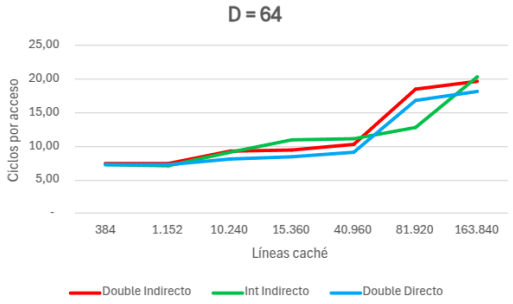

Figura 4: *Gráfica de resultados con D=64*

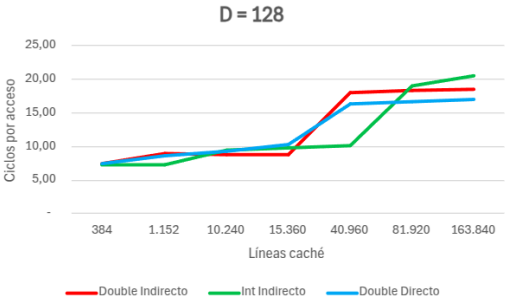

Figura 5: *Gráfica de resultados con D=128*

III. Conclusiones

Una vez realizado el experimento, podemos terminar con las conclusiones. Dado que D es un valor arbitrario y L se podría decir que también, el parámetro como tal es R. Con la fórmula que hemos razonado antes; $R = (8/D) * L$, se calcula este parámetro, que es el número de elementos de A sobre los que realizaremos la reducción. Sin embargo, debemos tener en cuenta el desplazamiento, por lo que el tamaño del vector no puede ser R, sino que será R*D. Entonces, como podemos ver, los valores que reciban D y L son de gran relevancia para calcular el tamaño del vector y el número de ciclos consumidos en cada acceso.

- En casos con localidad buena, como D = 1 o D = 4, el gran tamaño del vector es compensado con el número de aciertos que producirá la caché. Es decir, pongamos como ejemplo el caso de los `double`, si tenemos D = 1 y L = 163840, tendremos un R que será 8*163840. De la misma forma, el vector será de este tamaño debido a D = 1. Por lo que, tenemos un vector muy grande, sin embargo, tenemos prácticamente los mismos ciclos por acceso. Esto se debe a tener una localidad máxima, ya que el número de aciertos caché es mucho mayor que el número de fallos (7 a 1, asumiendo que solo trae 1 línea).
- En casos con una localidad mala, como son los otros tres valores de D, el tamaño del vector será mucho más relevante. Esto se debe a que en estos valores de desplazamiento llegamos a saltar varias líneas caché, produciendo rachas de muchos fallos consecutivos. Un ejemplo puede ser con D = 128. En el menor tamaño de vector, con L = 384, tenemos que R = L, por lo que el tamaño será 128*384, siendo este valor el número de elementos. Si lo dividimos entre 8 (los `double` por línea), obtenemos que ocupará 6144 líneas, por lo que cabe entre L1 y L2. Por el contrario, con L = 163840, el vector tendrá un número de elementos elevado; 128*163840. El número de líneas, por consecuencia, seguirá siendo elevado, tanto que no cabrá íntegramente en L1 y L2, produciendo muchos fallos por la localidad y, a su vez, retrasándose aún más debido a la necesidad de acceder al tercer nivel de caché o incluso a la memoria RAM.

A este factor de localidad espacial, también le debemos sumar la localidad temporal, ya que como comentamos, con una localidad óptima se cumplirá que si usamos un elemento, los elementos cercanos (localidad espacial) se usarán en breves (localidad temporal), por lo que un desplazamiento D = 1 cumple a la perfección, mientras que D = 128 no, ya que si usamos el elemento i-ésimo, el elemento i+1 no será usado. Por último, tenemos que tener en cuenta la técnica de *prefetching* que use la CPU que, a pesar de no haber encontrado una información concreta, el manual de optimizaciones de intel sí que nos informa de que la caché de la arquitectura *Ice Lake-SP* tiene una mejora en los *prefetchers* para aumentar el paralelismo de memoria.

Referencias

[1] INTEL CORPORATION, Ïntel® 64 and

IA-32 Architectures. Optimization Reference Manual: Volume 1", vol 1, Sec 2.4.1.3, pp 2-17, Çache and Memory Subsystem".