Localidad de los accesos a memoria en microprocesadores

Alex Baquero Domínguez, David Bernárdez Fernández

Arquitectura De Computadores
Grupo I
{alex.baquero,david.bernandez.fernandez}@rai.usc.es

Análisis del coste temporal por lectura (acceso) de datos, variando diferentes parámetros en el patrón de acceso y el tamaño del conjunto de datos. Considerando los conceptos de localidad espacial, temporal y precarga, se observará el efecto del sistema de memorias caché del microprocesador y se logrará interpretar el resultado de la localidad de las lecturas a memoria en las prestaciones de programas en microprocesadores.

***Palabras clave*— microprocesadores, memoria caché, jerarquía de memoria , tiempo de ejecución , acceso , localidad, precarga...**

I. Introducción

Este trabajo se basa en los estudios realizados a través de los experimentos mostrados en la sección III y sección IV en los cuales se pretende comprobar si dos patrones de acceso a los datos con diferente localidad consumen un tiempo de acceso a memoria diferente.

Estos estudios consisten en medir el tiempo de ejecución en número total de ciclos de procesador que necesitan las 10 repeticiones y a partir de él, calcular el número medio de ciclos por cada acceso a datos. Gracias a la precarga (prefetching) transparente de información en las cachés, este dato se estudiará e interpretará para entender el efecto del sistema de memorias caché del microprocesador al variar diferentes parámetros.

El objetivo es cambiar diferentes parámetros en el patrón de acceso y el tamaño del conjunto de datos, para caracterizar el coste temporal por entrada de datos y observar e explicar el efecto en nuestro sistema de memoria caché del microprocesador.

Esta representación se explicará gracias a los conceptos de localidad temporal, espacial y precarga (prefetching) y se conseguirá anotando en cada ejecución del programa el número de lecturas a memoria (variando el número de filas F y columnas C en cada caso)

Este informe queda estructurado de la siguiente manera:

En la sección II se dará a conocer la jerarquía de memoria caché del procesador utilizado en detalle. En los apartados III y IV se presentará el código de programación C de los ejercicios 1 y 2 respectivamente, y para concluir, en la sección V y VI se presentarán e interpretarán los resultados obtenidos en relación a los conceptos de localidad temporal y espacial en el acceso a los datos.

II. Características del procesador

En este apartado se conocerá la jerarquía de memoria caché del procesador en detalle, imprescindible para realizar los experimentos.

Se analizará el procesador que se utilizará en la experimentación, ya que los parámetros de esta se calcularán a partir de sus características. La información que se deberá conocer será el tamaño de las memorias caché, así como si estas son compartidas entre varios cores o no y el tamaño de líneas y palabras.

Hay diferentes posibles formas de averiguar esta información a través comandos en la terminal de linux:

- *hardinfo* permitirá conocer el tamaño de las distintas cachés, además de su tipo (asociativa por conjuntos en este caso) y el número de vías.
- *getconf -a,* con el cual se podrá visualizar entre otras cosas el tamaño de las líneas de cada memoria caché.

- *lscpu* se podrá obtener la arquitectura del procesador, dato con el que se podrá saber también el tamaño de palabra del sistema.

Después de ejecutar estos comandos, la información obtenida es la siguiente:

CUADRO 1
JERARQUÍA DE MEMORIA USADA

MODELO	INTEL CORE i5-8250u
NÚCLEOS	4
ARQUITECTURA	x86_64
PALABRAS	64 bits
NIVELES CACHÉ	3
L1	32KB
L2	256KB
L3	6144KB
TAMAÑO DE LÍNEA	64 bytes

Como se puede observar, en el **Cuadro 1**, el procesador de la máquina utilizada, es un Intel Core i5-8250u de 4 núcleos con una arquitectura x86_64 que utiliza palabras de 64 bits. Cuenta con 3 niveles de caché, el primero y segundo son propios de cada núcleo, mientras que el último es compartido por todos ellos. Los tamaños son de 32KB y 256KB para L1 y L2 (cada uno de los 4 núcleos) y 6144KB para la caché compartida L3.

III. CASO DE ESTUDIO 1

En este sección se presentará en que consiste el primer programa utilizado: se realizará un código en C que reserve memoria para una matriz de F filas y C columnas de manera dinámica a través de F punteros que tendrán a su vez un tamaño de C elementos double.

El objetivo será calcular la reducción del primer elemento de cada una de las filas de la matriz, para poder medir el coste medio de ciclos de cada acceso a memoria.

Para realizar estos experimentos se utilizarán unos valores de C concretos: 4, 8, 20 y 40. En el caso de F, se calcularán sus valores a partir de los tamaños de las memorias caché del procesador. Cada valor de F se obtendrá de modo que las lecturas a los valores de la matriz se hagan sobre L líneas caché diferentes. L tomará 8 valores relativos a S1, S2 y S3, que serán el número de líneas que caben en las cachés L1, L2 y L3 respectivamente.

El primer paso para obtener F, será conocer los valores de S1, S2 y S3. Para poder obtenerlos es necesario conocer tanto el tamaño de las líneas caché en el sistema, como el tamaño de las memorias caché (datos mostrados anteriormente). El número de líneas caché será igual a la división del tamaño total de estas entre el tamaño de sus líneas.

S1 -> líneas L1 = 32768 / 64 = 512 líneas por core

S2 -> líneas L2 = 262144 / 64 = 4096 líneas por core

S3 -> líneas L3 = 6291456 / 64 = 98 304, compartida por todos los cores

Los valores de L serán los siguientes:

CUADRO 2
VALORES DEL NÚMERO DE LÍNEAS CACHÉ QUE SE PRETENDE ACCEDER

FÓRMULA	L
0.5 * S1	*256*
1.5 * S1	*768*
0.5 * S2	*2048*
1.5 * S2	*6144*
0.5 * S3	*49152*
0.75 * S3	*73728*
2 * S3	*196608*
4 * S3	*393216*

A través de los valores de L calculados en el **Cuadro 2**, se conseguirá los valores de F.

Sabiendo que se accede solamente al primer elemento de cada fila y que el tamaño de una línea es de 64 bytes (equivalente al tamaño de 8 doubles) se obtendrá un valor de F distinto según el número de columnas.

En este caso en particular para C = 4, se accederá a una línea caché diferente cada dos filas, por lo que será necesario que los valores de F sean el doble de los valores de L. Los valores de F para C= 4 serán los mostrados en el **Cuadro 3**.

CUADRO 3
VALORES DEL NÚMERO DE FILAS DE LA MATRIZ

L	F
256	*512*
768	*1536*
2048	*4096*
6144	*12228*
49152	*98304*
73728	*147456*
196608	*393216*
393216	*786432*

Con C = 8, en una fila estará contenida exactamente una línea caché, mientras que con C = 20 serán 2 líneas y media y con C= 40, por en cada fila de la matriz se podrá acceder a 5 líneas caché diferente.

Como el acceso es solo al primer elemento de cada fila, en ninguno de estos casos la línea a la que se accede en una fila corresponde a la misma que a la que se accede en la siguiente. Por lo tanto, estos valores F se corresponderán con L.

A continuación se revisará el código del programa en C.

IMAGEN 1
FRAGMENTO DEL PROGRAMA EN C UTILIZADO PARA ESTUDIAR EL CASO DE ESTUDIO 1

arqComp1.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <pmmintrin.h>
#include <unistd.h>
#include <time.h>
#define F 6144
#define C 40//4, 8, 20, 40

```

En la **Imagen 1** se enseñan las librerías utilizadas para facilitar el cambio del código los parámetros F y C. Estos son constantes, las cuales el valor será modificado en cada ejecución.

El acceso a los datos de la matriz será aleatorio, para ello se utilizará un vector *indices*[] de tamaño F, que se generará de manera aleatoria, junto a otro auxiliar de tamaño F también. Además, se usará otro vector *red[]* de 10 elementos que será donde se almacenarán los valores que se obtengan de la reducción.

IMAGEN 2
FRAGMENTO DEL PROGRAMA EN C UTILIZADO PARA ESTUDIAR EL CASO DE ESTUDIO 1

```c

int ind[F]; //array de los índices
double red[10]; //vector para almacenar el resultado de cada ejecución

double **M; //: matriz de f punteros y c columnas

int main() {
    double ck;

    printf("F: %d\n", F);

    int ran, usados[F];

    srand(time(NULL)); //La semilla de aleatoriedad se establece según el tiempo

    M = _mm_malloc(F * C * 8, 64); //total de bytes, tamaño línea caché

    for (int i = 0; i < F; i++) {
        M[i] = (double*) malloc(C * sizeof (double));

        for (int j = 0; j < C; j++) {
            M[i][j] = rand();
        }
    }

    //Inicializar vector usados
    for (int i = 0; i < F; i++) {
        usados[i] = 0;
    }

    for (int i = 0; i < 10; i++) {
        red[i] = 0;
    }

```

El código de la **Imagen 2** corresponde tanto a la reserva de memoria de la matriz como de los vectores utilizados. En el caso de la matriz la función utilizada será *_mm_malloc* para alinear el bloque de memoria reservado con el inicio de una línea caché.

IMAGEN 3
FRAGMENTO DEL PROGRAMA EN C UTILIZADO PARA ESTUDIAR EL CASO DE ESTUDIO 1

```c
    //Calcular el vector indices
    for (int i = 0; i < F; i++) {
        ran = rand() % F;

        //calculamos números hasta que se llegue a una no usada
        while (usados[ran] != 0) {
            ran = rand() % F;
        }

        usados[ran] = 1; //marcamos la fila como usada
        ind[i] = ran;
    }

```

Para el cálculo de los índices aleatorios se empleará un vector auxiliar *usados*[], de forma que se irán introduciendo

índices de filas en cada interacción (siempre que no hayan sido ya usados). De este modo, la posición de ese índice aparece como un 1 en el vector auxiliar. Todo esto se puede ver en la **Imagen 3**.

IMAGEN 4
FRAGMENTO DEL PROGRAMA EN C UTILIZADO PARA ESTUDIAR EL CASO DE ESTUDIO 1

```
125       double n_accesos;
126
127       for (int i = 0; i < 10; i++) {
128
129           for (int j = 0; j < F; j++) {
130               red[i] += M[ind[j]][0];
131               n_accesos += 1;
132           }
133       }
134
```

En la **Imagen 4** se medirá el número de accesos.

El programa se ejecutará 10 veces para conseguir un resultado más uniforme en el número de ciclos. Con *n_accesos* se almacenará el número de accesos totales en las 10 ejecuciones, que después se dividirá para obtener el valor medio.

IMAGEN 5
FRAGMENTO DEL PROGRAMA EN C UTILIZADO PARA ESTUDIAR EL CASO DE ESTUDIO 1

```
135     ck = get_counter();
136
137     printf("\n Clocks=%1.10lf \n", ck);
138     printf("Ciclos por acceso: %1.10lf\n", ck/n_accesos);
139
140     /* Esta rutina imprime a frecuencia de reloxo estimada
141     coas rutinas start_counter/get_counter */
142     mhz(1, 1);
143
144     //Calcular media e imprimir valores
145     printf("Valores de red[]: \n");
146     double total = 0;
147
148     for (int i = 0; i < 10; i++) {
149         printf("%lf \n", red[i]);
150         total += red[i];
151     }
152
153     printf("\nMedia de los valores: %lf\n", total / 10.0);
154
155     _mm_free(M);
156
157     return 0;
158 }
```

Por último, se mostrarán los resultados en pantalla, además de los valores de *red[]* para evitar que el compilador realice optimizaciones decidiendo que los valores no se usan **(Imagen 5).**

IV. CASO DE ESTUDIO 2

Se presentará el planteamiento del apartado 2, así como el código realizado para su implementación.

El desarrollo será similar al del primer apartado. Se deberá reservar memoria para una matriz de F filas y C columnas y el objetivo será acceder a L líneas caché, pero en este caso la reducción será de los primeros elementos de cada línea caché, en vez de los primeros elementos de la fila.

Los valores de C y L serán los mismos que en el apartado anterior, sin embargo, los valores de F serán diferentes. En este caso, como se realizan accesos a los primeros elementos de cada línea caché, y no de cada fila, se deberán tener en cuenta el número de líneas que puede albergar una fila para realizar los cálculos.

Con estos valores se podrá evitar hacer comprobaciones del número de accesos realizado dentro del código de la reducción, pero a costa de esto, el número de accesos del experimento no se corresponde exactamente con el que se pretende. Se ha decidido hacer de esta forma, ya que los resultados serán más fieles a la realidad evitando la interferencia de los ciclos de las comprobaciones que podrían desvirtuar las medidas, teniendo como única desventaja un aumento muy pequeño en el número de accesos para algunos casos (nunca mayor de 4).

En el caso de C = 4 y C = 8 los valores coincidirán con los del apartado anterior. En C = 4 se accederá al principio de una línea caché cada dos filas, mientras que en C = 8 será en cada fila.

Para C = 20, se accederán a 5 líneas diferentes cada dos filas (3 accesos en la primera y dos en la segunda), por lo que F quedaría del siguiente modo (**Cuadro 4)**:

CUADRO 4
VALORES DEL NÚMERO DE FILAS DE LA MATRIZ DE C=20

L	F
256	*103*
768	*307*
2048	*819*
6144	*2458*
49152	*19661*
73728	*29491*
196608	*78643*
393216	*157287*

En el caso de C = 40, en cada fila se acceden a 5 líneas diferentes, por lo F será el resultado de dividir los valores de L entre 5 y redondear hacia arriba en caso de obtener un resultado decimal.

Los diferentes valores se indican en el **Cuadro 5.**

CUADRO 5
VALORES DEL NÚMERO DE FILAS DE LA MATRIZ DE C=40

L	F
256	*52*
768	*154*
2048	*410*
6144	*1229*
49152	*9831*
73728	*14746*
196608	*39322*
393216	*78644*

A continuación se realizará un análisis del código empleado para la realización de la experimentación. (mencionar que las librerías utilizadas son las mismas que en el apartado 1 y los parámetros C y F se han declarado como constantes)

IMAGEN 6
FRAGMENTO DEL PROGRAMA EN C UTILIZADO PARA ESTUDIAR EL CASO DE ESTUDIO 2

```
67
68  double red[10]; //vector para almacenar el resultado de cada ejecución
69  double **M; //: matriz de f punteros y c columnas
70
71  int main() {
72      double ck;
73      int ran, usados[F];
74
75      srand(time(NULL)); //La semilla de aleatoriedad se establece según el tiempo
76
77      M = _mm_malloc(F * C * 8, 64); //total de bytes, tamaño línea caché
78
79      for (int i = 0; i < F; i++) {
80          M[i] = (double*) malloc(C * sizeof (double));
81
82          for (int j = 0; j < C; j++) {
83              M[i][j] = rand();
84          }
85      }
86
87      for (int i = 0; i < 10; i++) {
88          red[i] = 0;
89      }
90
91
```

En la **Imagen 6** se puede apreciar un código similar al de la sección anterior. El principal cambio será la eliminación del vector índice y el vector usados debido a que se prescinde del acceso aleatorio.

IMAGEN 7
FRAGMENTO DEL PROGRAMA EN C UTILIZADO PARA ESTUDIAR EL CASO DE ESTUDIO 2

```
100
101     int indice = 0;
102     double n_accesos = 0;
103
104     for (int i = 0; i < 10; i++) {
105         indice = 0;
106
107         for (int j = 0; j < F; j++) { //recorre las filas
108             while(indice < (j+1)*C){
109                 red[i] += M[j][indice % C];
110                 indice += 8;
111                 n_accesos += 1;
112             }
113
114         }
115     }
116
```

En la **Imagen 7**, para poder acceder al primer elemento de cada fila se utilizará la variable índice que irá aumentando 8 unidades en cada interacción. Para usar al elemento que esté en el inicio de la línea caché será necesario el uso de la operación módulo respecto al número de columnas. Además, esta variable controlará el cambio de filas, de modo que cuando sea mayor que el número de la fila más una unidad (es decir, para la fila 0 el número será uno) multiplicado por el número de columnas se realizará el salto.

IMAGEN 8
FRAGMENTO DEL PROGRAMA EN C UTILIZADO PARA ESTUDIAR EL CASO DE ESTUDIO 2

```
118
119     ck = get_counter();
120
121     printf("\n Clocks=%1.10lf \n", ck);
122     printf("Ciclos por acceso a la mtriz = %1.10lf\n", ck/n_accesos);
123
124     /* Esta rutina imprime a frecuencia de
125      reloxo estimada coas rutinas start_counter/get_counter */
126     mhz(1, 1);
127
128     //Calcular media e imprimir valores
129     printf("Valores de red[]: \n");
130     double total = 0;
131
132     for (int i = 0; i < 10; i++) {
133         printf("%lf \n", red[i]);
134         total += red[i];
135     }
136
137     printf("\nMedia de los valores: %lf\n", total / 10.0);
138
139     _mm_free(M);
140
141     return 0;
142 }
143
```

Por último se volverán a realizar las impresiones de los elementos del array *red[]*, así como del número de ciclos

medios, además de liberar la memoria de la matriz. **(Imagen 8)**

V. Resultados

Se mostrará cómo afectan el valor del parámetro C y el número de líneas caché diferentes accedidas L al tiempo por acceso para el diferente caso de estudio.

Para ello, se seguirá el siguiente procedimiento en ambos casos de estudio:

1. Se medirá el tiempo de ejecución en número total de ciclos de procesador que necesitan las 10 repeticiones.
2. Se calculará el número medio de ciclos por cada acceso a datos, que será el dato que finalmente se interpretará.
3. Se repetirá el proceso que se acaba de indicar para cada par de valores de F y de C.
4. Para cada valor de número de columnas C (4,8,20,40), se realizará un total de 8 medidas para diferentes valores y número de filas F.
5. Para cada caso, se anotará en un excel para elaborar unas gráficas, necesarias para estudiar el efecto de localidad en los accesos a memoria.

En todos los experimentos realizados, se ha utilizado el sistema operativo Linux y el compilador gcc con la opción -O0 (no optimización). En ningún caso se ha abierto ningún programa, aparte del terminal, durante la realización del experimento.

A. Caso de estudio 1

Presentamos el fruto de los experimentos del estudio 1:

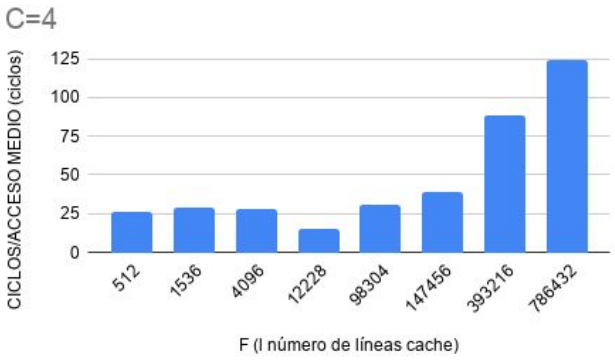
IMAGEN 9

ANOTACIÓN DE LAS DISTINTAS PRUEBAS REALIZADAS EN EL EXPERIMENTO 1

C	F	1 PRUEBA	2 PRUEBA	3 PRUEBA	4 PRUEBA	CICLOS/ACCESO MEDIO
4	512	25,64	25,61	25,6	25,83	25,67
	1536	28,44	27,58	31,92	27,5	28,86
	4096	32,27	24,38	33,01	20,94	27,65
	12228	14,42	15,85	16,66	14,99	15,48
	98304	30,64	30,94	30,17	31,83	30,895
	147456	38,65	39,08	37,94	38,78	38,6125
	393216	89,51	87,77	87,76	87,17	88,0525
	786432	120,39	124,31	125,35	125,62	123,9175
8	256	25,57	25,97	25,54	25,68	25,69
	768	27,08	31,65	31,75	31,23	30,4275
	2048	34,25	27,44	33,07	35,49	32,5625
	6144	14,22	13,22	13,53	15,66	14,1575
	49152	31,7	34,55	33,09	27,64	31,745
	73728	35,46	36,02	36,91	35,37	35,94
	196608	50,68	51,55	49,9	51,61	50,935
	393216	90,43	91,58	90,98	93,29	91,57
20	256	30,01	25,89	26,34	26,37	27,1525
	768	28,41	27,59	33,41	27,79	29,3
	2048	34,66	29,62	16,7	36,15	29,2825
	6144	16,17	15,08	17,21	16,64	16,275
	49152	27,72	35,22	27,82	30,35	30,2775
	73728	33,52	34,35	34,81	35,5	34,545
	196608	46,85	48,49	48,53	48,89	48,19
	393216	94,68	96,9	97,59	96,49	96,415
40	256	29,96	26,61	26,31	26,33	27,3025
	768	27,85	33,25	33,13	26,27	30,125
	2048	11,43	13,22	14,87	11,95	12,8675
	6144	15,43	14,53	14,68	61,94	26,645
	49152	29,02	25,05	27,54	29,97	27,895
	73728	34,43	35,9	34,1	34,08	34,6275
	196608	52,11	49,81	50,63	53,34	51,4725
	393216	104,23	108,93	99,83	99,8	103,1975

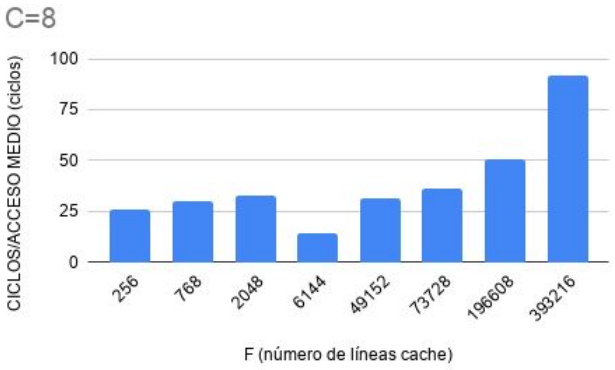
GRÁFICA 1

REPRESENTACIÓN GRÁFICA DEL ESTUDIO 1 PARA EL VALOR C=4 DEL COSTE EN CICLOS DE LECTURA (EJE Y) FRENTE AL NÚMERO TOTAL DE LÍNEAS CACHÉ DIFERENTES QUE SE REFERENCIAN (EJE X)



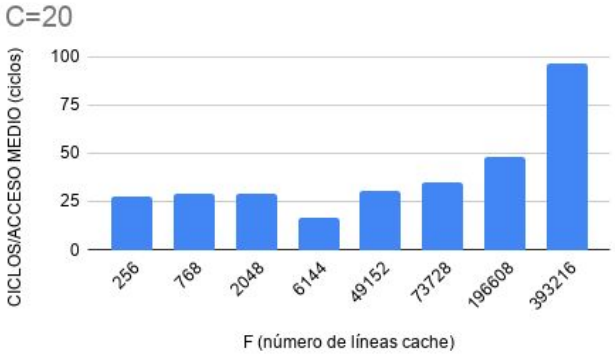
GRÁFICA 2

REPRESENTACIÓN GRÁFICA DEL ESTUDIO 1 PARA EL VALOR C=8 DEL COSTE EN CICLOS DE LECTURA (EJE Y) FRENTE AL NÚMERO TOTAL DE LÍNEAS CACHÉ DIFERENTES QUE SE REFERENCIAN (EJE X)



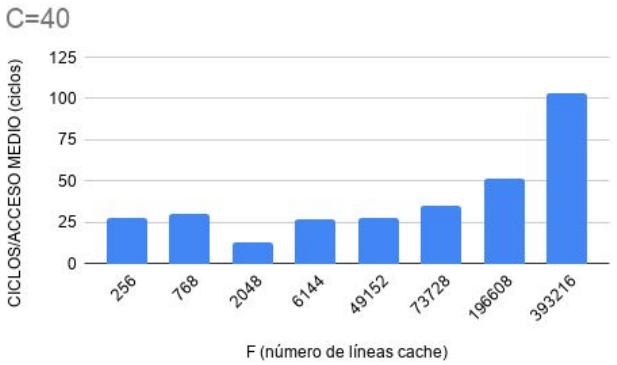
GRÁFICA 3

REPRESENTACIÓN GRÁFICA DEL ESTUDIO 1 PARA EL VALOR C=20 DEL COSTE EN CICLOS DE LECTURA (EJE Y) FRENTE AL NÚMERO TOTAL DE LÍNEAS CACHÉ DIFERENTES QUE SE REFERENCIAN (EJE X)



GRÁFICA 4

REPRESENTACIÓN GRÁFICA DEL ESTUDIO 1 PARA EL VALOR C=40 DEL COSTE EN CICLOS DE LECTURA (EJE Y) FRENTE AL NÚMERO TOTAL DE LÍNEAS CACHÉ DIFERENTES QUE SE REFERENCIAN (EJE X)



Se procederá a la interpretación de los resultados obtenidos en relación a los conceptos de localidad temporal y espacial en el acceso a los datos.

El funcionamiento de las memorias caché se basa en el principio de localidad: temporal (si usamos un dato o instrucción seguramente lo volveremos a usar en breve) y espacial (si usamos un dato o instrucción seguramente usaremos pronto otro situado en una dirección de memoria próxima). En principio, si un dato está en un nivel dado, tendrá copia en todos los niveles inferiores (L2,L3). Estos datos se mueven entre niveles adyacentes.

Como se puede ver en las anteriores gráficas (**Grafica 1, Grafica 2, Grafica 3, Grafica 4**), a medida que se aumenta el número de líneas caché, el número de ciclos de acceso a memoria va creciendo. Al aumentar el número de filas, aumentamos también el número de líneas caché a las que es necesario acceder, teniendo que buscarlas en los niveles inferiores. Esta necesidad provoca que aumente el número de ciclos por acceso de lectura pues será más costosa la operación (el tiempo de acceso aumenta en cada nivel de la jerarquía de memoria).

Los dos últimos valores aumentan de una manera más significativa. Este hecho se debe a que coinciden con 2*S3 y 4*S3: el número de accesos a memoria principal es mucho mayor que en otros casos, por lo que la penalización temporal es mucho mayor, ya que la diferencia de tiempo de acceso entre la caché y memoria principal es muy notable.

En el cuarto valor de F, existe un valor irregular que decrece de inmediato (en el caso de C=40, ocurre en el tercer valor). Esto puede deberse a que equivale a un tamaño concreto respecto a la caché L2, por lo que podría haber algún tipo de optimización.

En los microprocesadores se suele realizar precarga (prefetching) de datos en las cachés de manera transparente al usuario. La precarga es una técnica utilizada por los procesadores para aumentar el rendimiento de ejecución mediante la obtención de instrucciones o datos de su almacenamiento original en una memoria más lenta a una memoria local más rápida antes de que realmente se necesite. La precarga de datos en este caso sería a nivel de hardware, ya que la precarga de tipo software se evita con la opción de compilación -O0

B. CASO DE ESTUDIO 2

Presentamos el fruto de los experimentos del estudio 2:

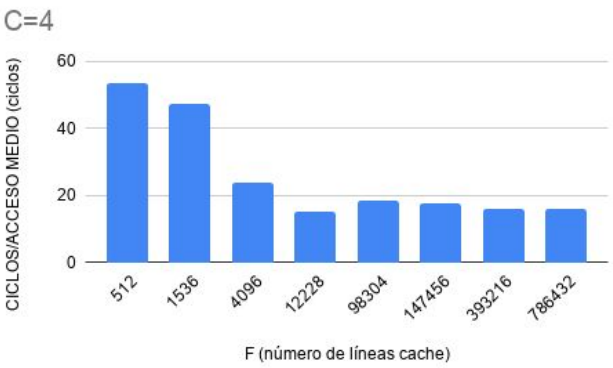
IMAGEN 10

ANOTACIÓN DE LAS DISTINTAS PRUEBAS REALIZADAS EN EL EXPERIMENTO 2

C	F	1 PRUEBA	2 PRUEBA	3 PRUEBA	4 PRUEBA	CICLOS/ACCESO MEDIO
4	512	53,48	53,42	60,38	46,1	53,345
	1536	47,24	47,14	47,16	47,27	47,2025
	4096	18,4	17,87	16,77	42,34	23,845
	12228	15,14	14,78	16,07	15,02	15,2525
	98304	19,7	15,9	22,17	16,9	18,6675
	147456	17,72	17,77	17,72	18,06	17,8175
	393216	16,16	15,91	16,05	16,22	16,085
	786432	15,98	16,07	16,09	16,09	16,0575
8	256	65,8	33,84	33,76	39,45	43,2125
	768	41,09	48,44	41,12	39,35	42,5
	2048	35,99	36,11	30,53	35,68	34,5775
	6144	13,19	11,46	11,58	11,54	11,9425
	49152	14,96	12,9	14,25	13,32	13,8575
	73728	13,89	14,84	15,37	13,05	14,2875
	196608	13,35	13,44	13,23	13,55	13,3925
	393216	12,59	12,67	12,72	12,76	12,685
20	103	59,86	36,36	42,17	36,77	43,79
	307	40,92	45,07	36,33	36,31	39,6575
	819	37,3	37,3	35,36	43,06	38,255
	2458	21,37	24,3	11,63	35,57	23,2175
	19661	13,91	14,6	12,42	14,65	13,895
	29491	14,2	14,43	13,28	15,04	14,2375
	78643	14,61	15,24	16,1	14,46	15,1025
	157287	14,84	14,81	14,44	13,89	14,495
40	52	46,47	34,98	34,7	45,8	40,4875
	154	35,12	29,89	39,71	29,95	33,6675
	410	34,81	36,64	34,47	29,11	33,7575
	1229	32,8	17,07	32,43	16,31	24,6525
	9831	38,94	12,98	12,24	13,53	19,4225
	14746	14,11	12,87	14,44	12,48	13,475
	39322	14,25	14,28	15,33	14,89	14,6875
	78644	12,68	14,13	14,08	12,34	13,3075

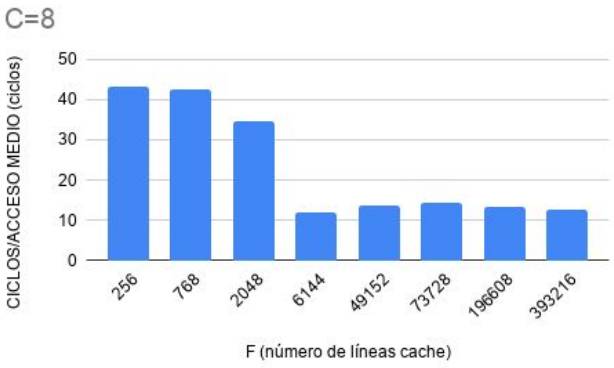
GRÁFICA 5

REPRESENTACIÓN GRÁFICA DEL ESTUDIO 2 PARA EL VALOR C=4 DEL COSTE EN CICLOS DE LECTURA (EJE Y) FRENTE AL NÚMERO TOTAL DE LÍNEAS CACHÉ DIFERENTES QUE SE REFERENCIAN (EJE X



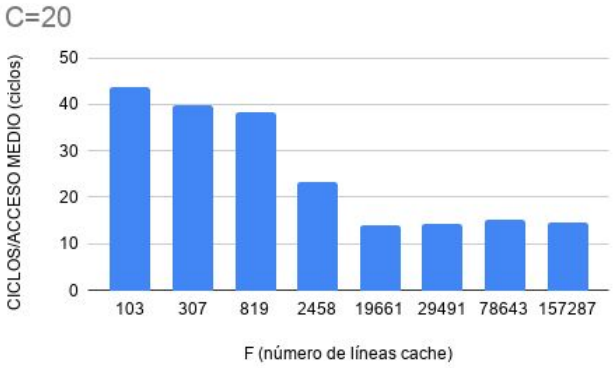
GRÁFICA 6

REPRESENTACIÓN GRÁFICA DEL ESTUDIO 2 PARA EL VALOR C=8 DEL COSTE EN CICLOS DE LECTURA (EJE Y) FRENTE AL NÚMERO TOTAL DE LÍNEAS CACHÉ DIFERENTES QUE SE REFERENCIAN (EJE X



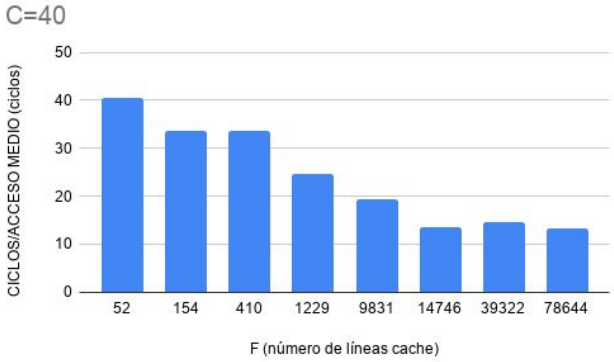
GRÁFICA 7

REPRESENTACIÓN GRÁFICA DEL ESTUDIO 2 PARA EL VALOR C=20 DEL COSTE EN CICLOS DE LECTURA (EJE Y) FRENTE AL NÚMERO TOTAL DE LÍNEAS CACHÉ DIFERENTES QUE SE REFERENCIAN (EJE X



GRÁFICA 8

REPRESENTACIÓN GRÁFICA DEL ESTUDIO 2 PARA EL VALOR C=40 DEL COSTE EN CICLOS DE LECTURA (EJE Y) FRENTE AL NÚMERO TOTAL DE LÍNEAS CACHÉ DIFERENTES QUE SE REFERENCIAN (EJE X



Con la ayuda de los conceptos de localidad secuencial y espacial en el acceso a los datos, se interpretará de los resultados obtenidos en el experimento 2.

Tal y como se ve la **Gráfica 5, Gráfica 6, Gráfica 7, Gráfica 8**, a medida que se aumenta el número de líneas caché, el número de ciclos de lectura a memoria va decreciendo.

En este caso la localidad presente en el programa es temporal en el caso del acceso a datos como en array red, y además está presente la localidad espacial, ya que se accede a datos que están cercanos en memoria.

Para explicar el decrecimiento presente en estas gráficas habría que volver a recurrir al concepto de precarga hardware. En este caso, para los valores más grandes el tiempo es menor ya que el procesador es capaz de detectar un patrón de acceso a los datos y precargar información que se necesitará en el futuro, mientras que en valores más pequeños no puede aprovecharse.

Otra explicación en la mejora de tiempos podría ser el mayor uso de la localidad espacial con valores grandes. Cuando se accede a un número de líneas caché inferior, los accesos serán caros debido a que hay que traer la información de la memoria principal. A medida que se accede a un mayor número de líneas caché, aumenta el aprovechamiento de la localidad secuencial, abaratando el acceso a líneas ya que han sido cargadas previamente.

VI. CONCLUSIONES

Se ha tratado de observar cómo se comporta el sistema de memorias caché del microprocesador al cambiar distintos parámetros y tamaños en la lectura de datos.

Con este fin, se han realizado unos programas en C que impriman el tiempo de ejecución en número total de ciclos de procesador que necesitan las 10 repeticiones, tras calcular el número medio de ciclos por cada acceso a datos y se ha ido anotando los resultados para luego ver su comportamiento en una serie de gráficas.

Se ha comprobado que el tiempo de acceso a memoria es diferente dado dos patrones de acceso de datos con diferentes localidades.

El principal problema que se ha ido encontrando al realizar el estudio es la anotación de resultados. Como se puede observar, en las gráficas no se visualizan perfectamente la curva de subida o bajada. Esto es consecuencia de que el microprocesador puede ejecutar diferentes procesos sin que sea apreciable para el usuario mientras se está realizando la experimentación.

El caso de estudio 1 permite comprobar que el acceso a la matriz con un patrón aleatorio provoca un aumento de ciclos de lectura al aumentar el número de líneas caché pues es necesario ir a buscar los datos en las cachés inferiores (operación costosa).

El caso de estudio 2 aprueba que el acceso secuencial a las líneas posibilita un mayor aprovechamiento de la localidad espacial, además de una mejora en el tiempo debido a la precarga hardware llevada a cabo por el procesador, lo que conlleva a la disminución del número de ciclos de lectura a memoria.

Finalmente, exponer que este estudio sería posible completarlo en un futuro trabajo a través de Simula3MS (realizado en la Universidad de A Coruña) que implementa un subconjunto de instrucciones basadas en el repertorio de instrucciones del procesador MIPS R2000/R3000. Este programa se suministra de forma libre, se permite su uso, modificación y distribución, pero no su apropiación para crear software propietario, secreto y no accesible a la comunidad.

Referencias

[1] Patterson, David. A. y Hennessy, John .L., Computer Organization and Design ARM Edition: The Hardware Software Interface, 4 Edición, USA, Morgan Kaufmann, 2017, 978- 0128017333