
# CIENCIA DE LA COMPUTACIÓN

# INFORME CACHE AND PROGRAMS

# COMPUTACIÓN PARALELA Y DISTRIBUIDA

NICOLE DURAND ZEBALLOS

SEMESTRE VIII

2022

“La alumna declara haber realizado el presente trabajo de acuerdo a las normas de la Universidad Católica San Pablo”

MEMORIA CACHE:
La memoria caché es una memoria pequeña y rápida que se interpone entre la CPU y la memoria principal para que el conjunto opere a mayor velocidad. Para ello es necesario mantener en la caché aquellas zonas de la memoria principal con mayor probabilidad de ser referenciadas. Esto es posible gracias a la propiedad de localidad de referencia de los programas.

Una regla empírica que se suele cumplir en la mayoría de los programas revela que gastan el 90% de su tiempo de ejecución sobre sólo el 10% de su código que es donde se hace el uso de memoria.

Localidad temporal: las palabras de memoria accedidas recientemente tienen una alta probabilidad de volver a ser accedidas en el futuro cercano. La localidad temporal de los programas viene motivada principalmente por la existencia de bucles.

- ➔ Localidad temporal: Los datos en el espacio de memoria a las recientemente referenciadas tienen una alta probabilidad de ser también referenciadas en el futuro cercano. Es decir, que los datos próximos en memoria en memoria tienden a ser referenciadas juntas en el tiempo.
- ➔ La localidad espacial viene motivada fundamentalmente por la linealidad de los programas (secuenciamiento lineal de las instrucciones) y el acceso a las estructuras de datos regulares.

Cuando una dirección se presenta en el sistema caché pueden ocurrir dos cosas:
- ➔ Cache Hit: el contenido de la dirección se encuentre en un bloque ubicado en una línea de la caché.
- ➔ Cache miss: el contenido de la dirección no se encuentre en ningún bloque ubicado en alguna línea de la caché.

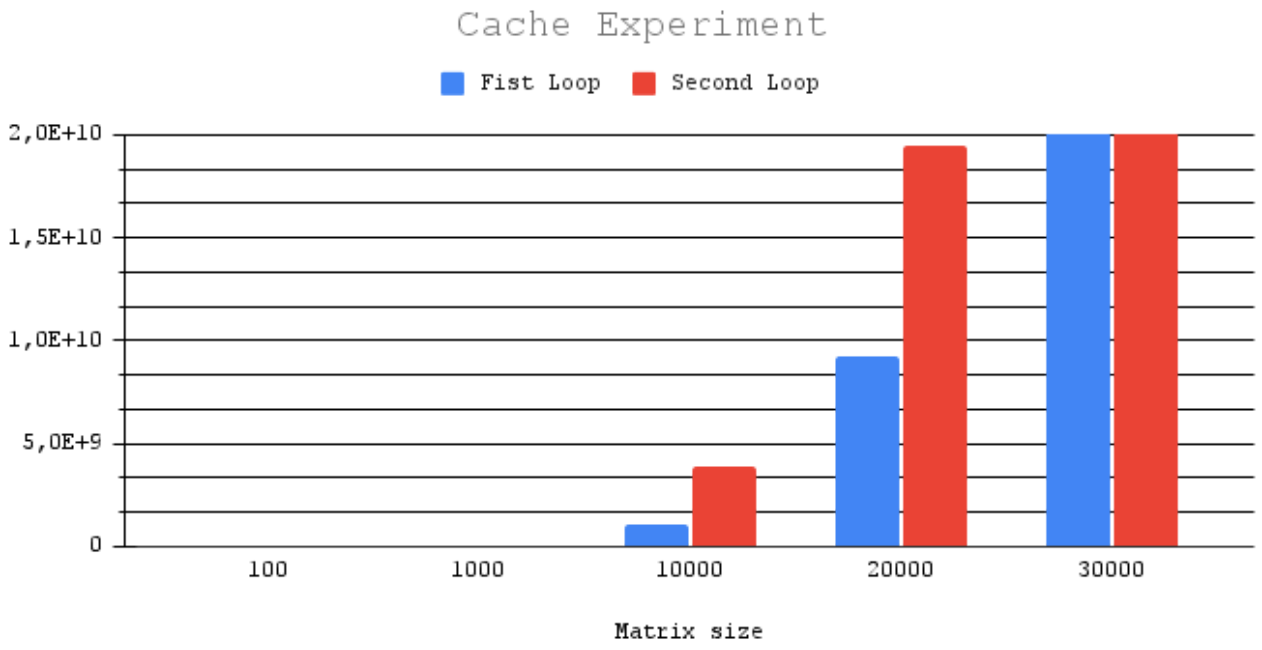
Si en la ejecución de un programa se realizan Nr referencias a memoria, de las que Na son aciertos caché y Nf fallos caché, se definen los siguientes valores:

Tasa de aciertos: Ta = Na/ Nr
Tasa de fallos: Tf = Nf/ Nr

PARA EL EXPERIMENTO:
Se hicieron pruebas en una matriz de tamaño n*n, donde se n tomo el valor 100,1000,10000,20000,30000 datos respectivamente, y se itero dentro de la matriz de dos formas distintas.

Obteniendo así los siguientes resultados:

```
------------ 100 Items in matrix ------------
** First Loop Time ->
136080
** Second Loop Time ->
126623
------------ 1000 Items in matrix ------------
** First Loop Time ->
11141793
** Second Loop Time ->
22069279
------------ 10000 Items in matrix ------------
** First Loop Time ->
1055572215
** Second Loop Time ->
3923835014
------------ 20000 Items in matrix ------------
** First Loop Time ->
9262042643
** Second Loop Time ->
19432529401
------------ 30000 Items in matrix ------------
** First Loop Time ->
34942002483
** Second Loop Time ->
160981288789
Program ended with exit code: 0
```



La primera forma de iterar la matriz fue por filas, y esta presenta mejores tiempos de ejecución, pues como se puede observar en la imagen a continuación los datos necesarios para los cálculos ya estarian previamente cargados en el cache teniendo así mayor cantidad de cache hits por lo tanto una mayor tasa de aciertos y se encontraron cerca uno de otro por lo que tomaría menos tiempo encontrarlos.

| Cache Line | Elements of A | | | |
|---|---|---|---|---|
| 0 | A[0][0] | A[0][1] | A[0][2] | A[0][3] |
| 1 | A[1][0] | A[1][1] | A[1][2] | A[1][3] |
| 2 | A[2][0] | A[2][1] | A[2][2] | A[2][3] |
| 3 | A[3][0] | A[3][1] | A[3][2] | A[3][3] |

Y para la segunda forma de iterar que sería por columnas, se obtuvieron tiempos de ejecución mayores, pese a que se está realizando en sí la misma acción solo que en un orden diferente, pues los datos no se encuentran tan juntos, dando mayor cantidad de cache misses por lo que tomaría más tiempo buscar el dato que se necesita y teniendo una tasa de fallos más alta.

| Cache Line | Elements of A | | | |
|---|---|---|---|---|
| 0 | A[0][0] | A[0][1] | A[0][2] | A[0][3] |
| 1 | A[1][0] | A[1][1] | A[1][2] | A[1][3] |
| 2 | A[2][0] | A[2][1] | A[2][2] | A[2][3] |
| 3 | A[3][0] | A[3][1] | A[3][2] | A[3][3] |

En conclusión, considero que este experimento es muy útil para poder darnos cuenta que al programar no solo se debe pensar en llegar a un resultado final sino el cómo llegamos a este también y tener en cuenta que el uso correcto del hardware puede ayudarnos a obtener un mejor y más eficiente resultado, por lo que en programas futuros debemos tenerlo en cuenta desde la implementación del código.

Bibliografía:

*Fdi.ucm.es*, 2022. [Online]. Available: https://www.fdi.ucm.es/profesor/jjruz/WEB2/Temas/EC6.pdf. [Accessed: 24- Aug- 2022].

Github:
https://github.com/nicoleeed/Computacion-Paralela-y-Distribuida/tree/main/Cache%20and%20Programs