Clase del día - 02/03/2021

La clase de hoy vamos a ver el tema de jerarquía de memoria y vamos a estudiar dos conceptos muy importantes relacionados con la cache: la localidad espacial y la localidad temporal.

**Jerarquía de memoria**

La jerarquía de memoria puede verse como una pirámide dónde cada nivel representa una capa de hardware que almacena datos.

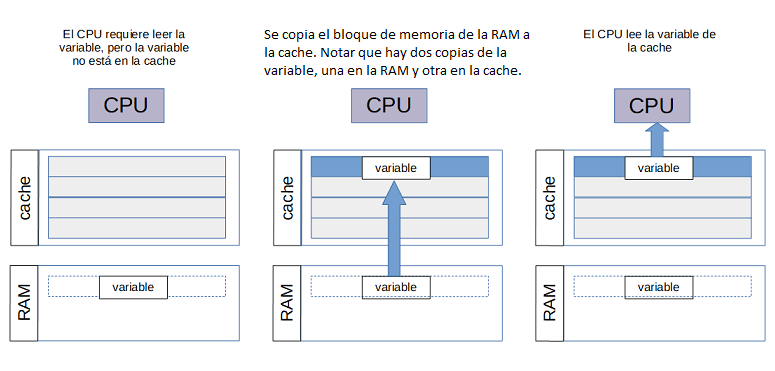


1. El CPU utiliza los **registros**para realizar operaciones aritméticas, lógicas y de control.
2. La memoria **cache**consiste en una memoria asociativa. Las memorias asociativas son muy rápidas, pero como son costosas, suelen ser de poca capacidad. La memoria cache puede estar dividida en varios niveles L1, L2, ...
3. La **memoria** **RAM** (Random Access Memory) suele ser es una memoria dinámica, por lo que requiere tener alimentación eléctrica constante para conservar los datos. Para escribir o leer una localidad de memoria en la RAM es necesario indicar la dirección de la localidad.
4. El **disco duro** almacena de manera persistente grandes cantidades de datos.
5. Los **respaldos** pueden ser discos duros de gran capacidad, discos ópticos, cintas, entre otros.

**La memoria cache**

Lectura de una variable

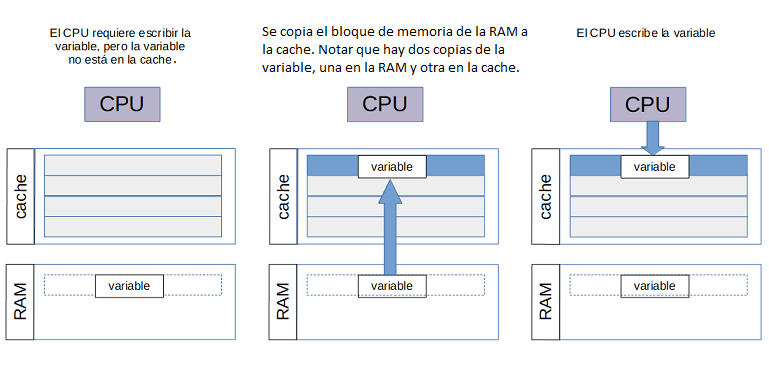
Cuando el CPU requiere leer una variable que se encuentra en la memoria RAM, busca la variable en la cache, si la variable no existe en la cache, copia el bloque de datos (que contiene a la variable) a la cache, entonces el CPU lee la variable de la cache.



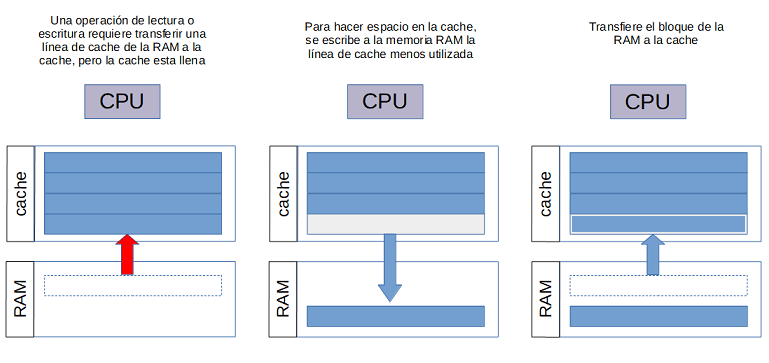
Al bloque de memoria que se transfiere de la RAM a la cache se le llama **línea de cache**. El tamaño de una línea de cache típicamente es de cientos de Kilobytes o Megabytes.

Escritura de una variable

Por otra parte, si el CPU requiere escribir una variable, busca la variable en la memoria cache, si existe, escribe el valor de la variable en la cache, si no existe, entonces copia la línea de cache (que contiene la variable) de la memoria RAM a la cache, y luego escribe el valor de la variable que está en la cache.



Debido a que la cache tiene un tamaño limitado (del orden de Megabytes), eventualmente se llenará. Para liberar líneas, la cache escribe a la memoria RAM las líneas menos utilizadas.



Como podemos ver, el CPU nunca lee o escribe datos directamente a la memoria RAM.

Así mismo, la cache nunca lee o escribe variables individuales a la memoria RAM, sino que siempre la transferencia de datos entre la cache y la memoria RAM se realiza en bloques (líneas de cache).

**Localidad espacial y localidad temporal**

Supongamos que el jefe de Recursos Humanos de una empresa le pide a su asistente los expedientes de algunos empleados en diferentes momentos del día.

Los expedientes se encuentran almacenados en cajas en el archivo de personal.

Cada caja contiene los expedientes organizados por apellido paterno, es decir, una caja contiene los expedientes de los empleados cuyo apellido paterno inicia con "A", otra caja contiene los expedientes de los empleados cuyo apellido paterno inicia con "B", y así sucesivamente.

La asistente puede ir al archivo de personal a traer un expediente o traer una caja completa.

En términos computacionales:

* Los expedientes representan los datos que se transfieren de la memoria RAM a la cache.
* El archivo dónde se encuentran los expedientes representa la memoria RAM.
* Una caja de expedientes representa una línea de cache.
* El jefe representa el CPU.

Consideremos tres casos:

Caso 1. La asistente obtiene expedientes individuales del archivo.

1. El jefe le pide a su asistente el expediente del Sr. González.
2. La asistente va al archivo a traer el expediente del Sr. González.
3. La asistente le da a su jefe el expediente del Sr. González
4. El jefe le pide a su asistente el expediente del Sr. Gómez.
5. La asistente va al archivo a traer el expediente del Sr. Gómez.
6. La asistente le da a su jefe el expediente del Sr. Gómez.
7. El jefe regresa a su asistente el expediente del Sr. González.
8. La asistente va al archivo a dejar el expediente del Sr. González
9. El jefe le pide a su asistente el expediente del Sr. González.
10. La asistente va al archivo a traer el expediente del Sr. González.
11. La asistente le da a su jefe el expediente del Sr. González.



Entonces la asistente tiene que ir cuatro veces al archivo.

Caso 2. La asistente obtiene cajas de expedientes del archivo.

1. El jefe le pide a su asistente el expediente del Sr. González.
2. La asistente va al archivo a traer la caja correspondiente a la letra ”G”.
3. La asistente le da a su jefe el expediente del Sr. González.
4. El jefe le pide a su asistente el expediente del Sr. Gómez.
5. La asistente le da a su jefe el expediente del Sr. Gómez.
6. El jefe regresa el expediente del Sr. González.
7. El jefe le pide a su asistente el expediente del Sr. González.
8. La asistente le da a su jefe el expediente del Sr. González.



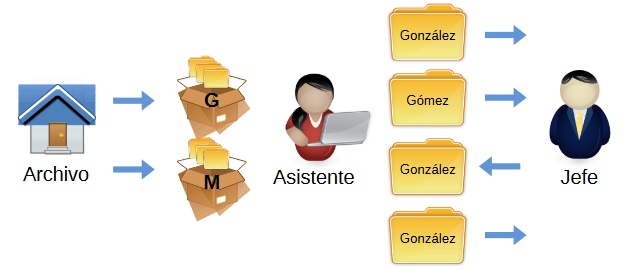
Entonces la asistente sólo va una vez al archivo. Los expedientes solicitados por el jefe se encuentran en la misma caja. El jefe pide más de una vez el expediente del Sr. González el mismo día.

La asistente ha descubierto los conceptos de localidad espacial y localidad temporal.

* Los datos presentan **localidad espacial** si al acceder un dato existe una elevada probabilidad de que datos cercanos sean accedidos poco tiempo después. En el ejemplo, los expedientes solicitados por el jefe presentan localidad espacial ya que se encuentran en la misma caja (digamos, la misma línea de cache).
* Un dato presenta **localidad temporal** si después de acceder el dato existe una elevada probabilidad de que el mismo dato sea accedido poco tiempo después. En el ejemplo, el expediente del Sr. González presenta localidad temporal ya que el jefe lo pide más de una vez el mismo día.

Caso 3. La asistente obtiene cajas de expedientes del archivo.

1. El jefe le pide a su asistente el expediente del Sr. González.
2. La asistente va al archivo a traer la caja correspondiente a la letra “G”.
3. La asistente le da a su jefe el expediente del Sr. González.
4. El jefe le pide a su asistente el expediente del Sr. Morales.
5. La asistente va al archivo a traer la caja correspondiente a la letra “M”.
6. La asistente le da a su jefe el expediente del Sr. Morales.
7. El jefe regresa el expediente del Sr. González.
8. El jefe le pide a su asistente el expediente del Sr. González.
9. La asistente le da a su jefe el expediente del Sr. González.



Entonces la asistente tiene que ir dos veces al archivo. Los expedientes del Sr. González y del Sr. Morales no presentan localidad espacial ya que se encuentran en diferentes cajas. El expediente del Sr. González presenta localidad temporal ya que el jefe lo pide más de una vez el mismo día.

Analicemos qué pasa en cada caso:

* Caso 1. En las primeras computadoras no había cache, por tanto el CPU accedia directamente los datos en la memoria. Debido a que la memoria era muy lenta, el CPU tenía que esperar mucho tiempo a que se leyera y/o escribieran los datos en la memoria RAM.
* Caso 2. La cache intercambia bloques de datos con la RAM. Dado que los datos presentan localidad espacial y localidad temporal, se reduce substancialmente los accesos a la memoria RAM, lo cual aumenta la eficiencia del programa ya que la RAM es una memoria lenta comparada con la cache.
* Caso 3. Los datos no presentan localidad espacial por tanto la cache transfiere bloques completos cada vez que se requiere leer o escribir un dato. En este caso tener la cache resulta más ineficiente que no tenerla (como sería en el caso 1).

La conclusión a la que llegamos es la siguiente: **la cache solo es de utilidad cuando los datos presentan localidad espacial y/o localidad temporal**.

Sin embargo, la cache no se puede "apagar", por tanto es necesario saber programar para la cache, o en otras palabras, es necesario que los programas presenten la máxima localidad espacial y/o localidad temporal.

Ahora veremos un ejemplo de cómo programar tomando en cuenta la cache.

**Caso de estudio: Multiplicación de matrices**

Como se explicó anteriormente, la cache acelera el acceso a los datos que presentan localidad espacial y/o localidad temporal, sin embargo no siempre los algoritmos están diseñados para acceder a los datos de manera que se privilegie el acceso a la memoria en forma secuencia (localidad espacial)  Vs. el acceso a la memoria en forma dispersa.

El siguiente programa multiplica dos matrices cuadradas A y B utilizando el algoritmo estándar (renglón por columna), en este caso las matrices tienen un tamaño de 1000x1000:

class MultiplicaMatriz  
{  
  static int N = 1000;  
  static int[][] A = new int[N][N];  
  static int[][] B = new int[N][N];  
  static int[][] C = new int[N][N];

  public static void main(String[] args)  
  {  
    long t1 = System.currentTimeMillis();  
  
    // inicializa las matrices A y B

    for (int i = 0; i < N; i++)  
      for (int j = 0; j < N; j++)  
      {  
        A[i][j] = 2 \* i - j;  
        B[i][j] = i + 2 \* j;  
        C[i][j] = 0;  
      }  
  
    // multiplica la matriz A y la matriz B, el resultado queda en la matriz C  
  
    for (int i = 0; i < N; i++)  
      for (int j = 0; j < N; j++)  
        for (int k = 0; k < N; k++)  
           C[i][j] += A[i][k] \* B[k][j];  
  
    long t2 = System.currentTimeMillis();

    System.out.println("Tiempo: " + (t2 - t1) + "ms");

  }  
}

Es necesario tomar en cuenta que Java almacena las matrices en la memoria como renglones, por lo que el acceso a la matriz B (por columna) es muy ineficiente si las matrices son muy grandes, ya que cada vez que se accede un elemento de la matriz B, se transfiere una línea de cache completa de la RAM a la cache.

El acceso a la matriz A es muy eficiente debido a que los elementos de la matriz A se leen secuencialmente, es decir, el acceso es por renglón, tal como la matriz se encuentra almacenada en la memoria.

Ahora vamos a modificar el algoritmo de multiplicación de matrices de manera que incrementemos la localidad espacial haciendo que el acceso a la matriz B sea por renglones y no por columnas.

El cambio es muy simple, solamente necesitamos intercambiar los índices que usamos para acceder los elementos de la matriz B, la cual previamente hemos transpuesto (es necesario transponer la matriz B para que el algoritmo siga calculando el producto de las matrices).

class MultiplicaMatriz\_2  
{  
  static int N = 1000;  
  static int[][] A = new int[N][N];  
  static int[][] B = new int[N][N];  
  static int[][] C = new int[N][N];

  public static void main(String[] args)  
  {  
    long t1 = System.currentTimeMillis();  
  
    // inicializa las matrices A y B  
  
    for (int i = 0; i < N; i++)  
      for (int j = 0; j < N; j++)  
      {  
        A[i][j] = 2 \* i - j;  
        B[i][j] = i + 2 \* j;  
        C[i][j] = 0;  
      }  
  
    // transpone la matriz B, la matriz traspuesta queda en B  
  
    for (int i = 0; i < N; i++)  
      for (int j = 0; j < i; j++)  
      {  
        int x = B[i][j];  
        B[i][j] = B[j][i];  
        B[j][i] = x;  
      }  
  
    // multiplica la matriz A y la matriz B, el resultado queda en la matriz C  
    // notar que los indices de la matriz B se han intercambiado  
  
    for (int i = 0; i < N; i++)  
      for (int j = 0; j < N; j++)  
        for (int k = 0; k < N; k++)  
           C[i][j] += A[i][k] \* B[j][k];  
  
    long t2 = System.currentTimeMillis();  
    System.out.println("Tiempo: " + (t2 - t1) + "ms");  
  }  
}

El resultado es un acceso más eficiente a los elementos de la matriz B, debido a que ahora se leen los elementos de B en forma secuencial, lo cual aumenta la localidad espacial y temporal de los datos.

Al ejecutar los programas MultiplicaMatriz.java y MultiplicaMatriz\_2.java para diferentes tamaños de las matrices, se puede observar que el algoritmo que accede ambas matrices por renglones (el segundo programa) es mucho más eficiente, ya que en éste algoritmo la localidad espacial y la localidad temporal de los datos es mayor debido a que las matrices A y B son accedidas por renglones, tal como se almacenan en la memoria por Java.

Actividades individuales a realizar

1. Compilar y ejecutar los programas MultiplicaMatriz.java y MultiplicaMatriz\_2.java.
2. ¿Por qué el segundo programa es más rápido que el primero?
3. ¿Podría plantear otro programa dónde el aumento de la localidad espacial y/o temporal hace más eficiente la ejecución?