Teoria 2- Memory wall

miércoles, 26 de marzo de 2025 10:53

- Limitaciones del sistema de memoria, no la velocidad del procesador
- Los parámetros del sistema importantes son latencia y ancho de banda
  - Latencia es el tiempo que pasa desde que solicitas el dato hasta que está
  - Ancho de banda es la velocidad con la que el sistema alimenta al proce
- Podemos achicar la latencia usando cache: memoria intermedia entre registros de cpu y memoria principal

~~Las matrtices en general se almacenan por fila en memoria. Si yo al operarla la recorro por columnas, hago banda de saltos en memoria, no me conviene.~~

Columnas:

```
for (i = 0; i < 1000; i++) {
        suma[i] = 0.0;
        for (j = 0; j < 1000; j++)  suma[i] += B[j][i];
}
```

Filas:

```
for (i = 0; i < 1000; i++) suma[i] = 0.0;
for (j = 0; j < 1000; j++)
        for (i = 0; i < 1000; i++)
                suma[i] += B[j][i];
```

Recordar: el acceso es [fiila, columna]

- **Alternativa 1 (arreglo estático):**

```
#define N 100

int main (int argc, char * argv[]
     …
     float matriz[N][N];
     …
}
```

¿Cómo accedo a la posición [i , j] siendo *i* el número de fila y *j* el de la columna?

```
matriz[i][j];
```

- ¿Tiene un tamaño máximo por la forma en que está declarado? ✗
- ¿Puedo cambiar su tamaño en ejecución? ✗
- ¿Puedo elegir cómo se organizan sus datos (por filas o por columnas)? ✗
- ¿Están todos sus datos contiguos en la memoria? ✓

Definición de arreglos en C

- **Alternativa 3 (arreglo dinámico como vector de punteros a filas/columnas):**

```
#define N 100

int main (int argc, char * argv[])
     …
     float ** matriz = malloc(N*sizeof(float*));
     for (i=0; i < N; i++)
          matriz[i] = mallo
     …
}
```

¿Cómo accedo a la posición [i , j] siendo *i* el número de fila y *j* el de la columna?

```
Por filas: matriz[i][j];
Por columnas: matriz[j][i];
```

- ¿Tiene un tamaño máximo por la forma en que está declarado? ✓
- ¿Puedo cambiar su tamaño en ejecución? ✓
- ¿Puedo elegir cómo se organizan sus datos (por filas o por columnas)? ✓
- ¿Están todos sus datos contiguos en la memoria? ✗

Eso de por filas/ columans es medio mentira pq accedes a distintos datos accediendo así

- **Alternativa 4 (arreglo dinámico como vector de elementos):**

```
#define N 100

int main (int argc, char * argv[]) {
    ...
    float * matriz = malloc(N*N*sizeof(float));
    ...
}
```

- ¿Tiene un tamaño máximo por la forma en que está declarado? ✅
- ¿Puedo cambiar su tamaño en ejecución? ✅
- ¿Puedo elegir cómo se organizan sus datos (por filas o por columnas)? ✅
- ¿Están todos sus datos contiguos en la memoria? ✅

- Cuando una matriz está organizada por filas, se debe multiplicar el número de fila por la cantidad de elementos de cada fila (cantidad de columnas) y sumarle el número de columna.
  - En el ejemplo anterior: `matriz [i*N+j]`

- Cuando una matriz está organizada por columnas, se debe multiplicar el número de columna por la cantidad de elementos de cada columna (cantidad de filas) y sumarle el número de fila.
  - En el ejemplo anterior: `matriz [j*N+i]`

En este caso, en lugar de usar un **arreglo de punteros a filas**, la matriz se almacena como **un único vector lineal** en memoria.

c — Copy — Edit

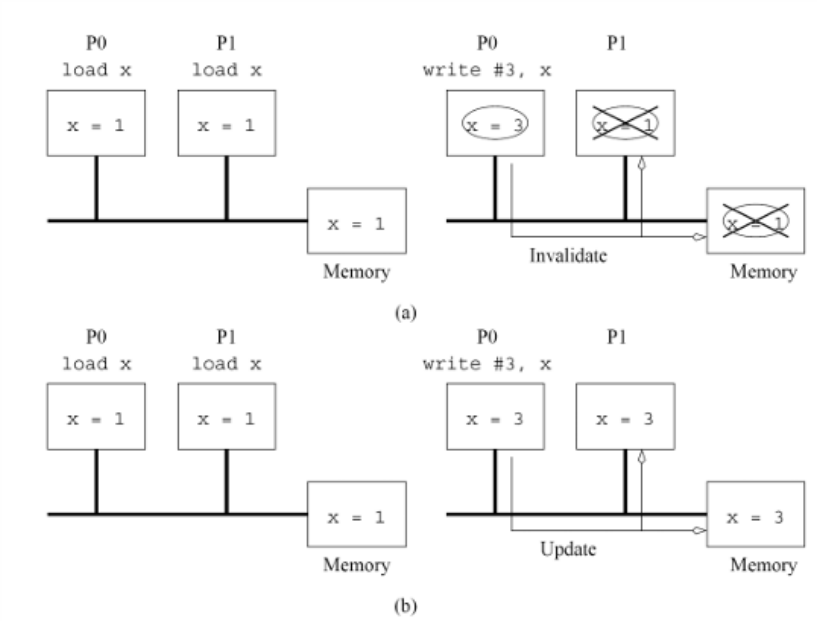
```c
#define N 100
int main (int argc, char * argv[]) {
    float *matriz = malloc(N * N * sizeof(float));
}
```

Esto crea un arreglo de `N × N` **elementos en una sola dimensión**, es decir, **una matriz almacenada en un solo bloque de memoria.**

Coherencia de cache en arquitecturas multiprocesador:
Cuando hay memoria compartida, pueden haber varias copias de lo mismo. Hay que tener cpmsostemncia entre copias

El mecanismo de coherencia debe asegurar que todas las operaciones realizadas sobre las múltiples copias son serializables: tiene que existir algún orden de ejecución secuencial que se corresponde con la planificación paralela

Protocolos de coherencia de caché: (a) invalidación (b) actualización

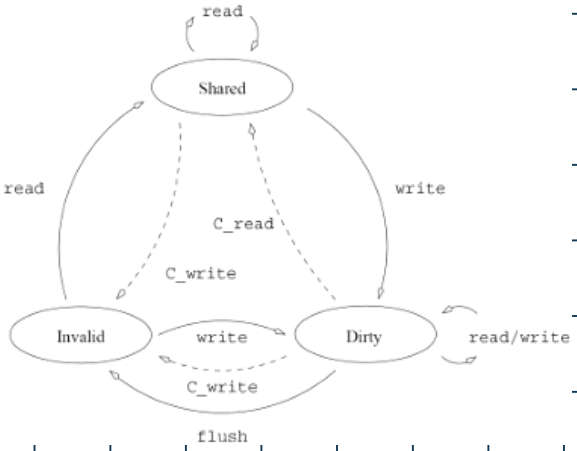


- Si un procesador lee un dato una vez y no vuelve a usarlo, un protocolo de actualización puede generar un overhead innecesario
  En este caso es mejor un protocolo de invalidación
- Si dos procesadores trabajan sobre la misma variable en forma alternada, un protocolo de actualización será mejor opción
  Se evita/reduce ocio por espera del dato actualizado

  Relación costo-beneficio: los protocolos de actualización pueden producir overhead por comunicaciones innecesarias mientras que los de invalidación pueden producir ocio ante la espera de actualización de un dato

Protocolos de coherencia de caché basados en invalidación

- Esquema simple donde cada copia se asocia con uno de 3 estados: *compartida (shared)*, *inválida (invalid)* o *sucia* (*dirty)*
- En el estado *compartida*, hay múltiples copias válidas del dato (en diferentes memorias). Ante una escritura, pasa a estado *sucia* donde se produjo mientras que el resto se marca como *inválida.*
- En el estado *sucia*, la copia es válida y se trabaja con esta.
- En el estado *inválida*, la copia no es válida. Ante una lectura, se actualiza a partir de la copia válida (la que está en estado *sucia*)



Mecanismos para la coherencia de cache

Sistemas snoopy:

- sistemas multiprocesador interconectados con alguna red broadcast, como bus o anillo
- caché de cada procesador mantiene un conjunto de tags asociados a sus bloques, los cuales determinan su estado.
- Todos los procesadores monitorizan (snoop) el bus, lo que permite realizar las transiciones de estado de sus bloques:
  - Cuando el hw snoopy detecta una lectura sobre un bloque de caché marcado como sucio, entonces toma el control del bus y cumple el pedido.
  - Cuando el hw snoopy detecta una escritura sobre un bloque de datos del cual tiene copia, entonces la marca como inválida

Me quedé en la diapo 38, falta un cachito :p