Procesadores Comerciales (3º Grado Informática)

Arq**T**ec**C**om - Dpto. Informática e Ingeniería de Sistemas.

CPS – UniZar

Práctica 5: Simulador de una cache de datos asociativa

Los objetivos de esta práctica son: i) entender y modelar una memoria cache asociativa por conjuntos a nivel de respuesta acierto/fallo, ii) estudiar el comportamiento y cuantificar las prestaciones de la cache ante patrones de acceso con distinta localidad temporal/espacial.

1. Implementación del simulador de memoria cache

En esta práctica se define y posteriormente se implementa un simulador de una cache de 8 KBytes, con bloques de 64 bytes, asociatividad 2 y reemplazo LRU. En escritura en fallo usa la política convencional (fetch-on-write-miss y write-allocate). En escritura en acierto usa write-back. Las estructuras de datos necesarias se definen a continuación, y estarán disponibles en el fichero `/export/home/alumnos/a01/cache/cache.c`:

```
#define SETS 64
#define ASOC 2

typedef struct CACHEblock {
    unsigned long long direccion;
    char valido;
    char sucio;         /* en valido y sucio solo se usa un bit */
};

typedef struct CACHEset {
    struct CACHEblock bloques[ASOC];
    char LRU;           /* en LRU solo se usa un bit para ASOC=2 */
};

struct CACHEset Cache_datos[SETS];
```

Definiremos dos funciones que añadiremos al fichero cache.c: leer() y escribir()

Ambas reciben como parámetro una dirección de memoria y devuelven un booleano que indica acierto/fallo

```
char leer(unsigned long long direccion);

char escribir(unsigned long long direccion);
```

Las funciones implementan el acceso a cache y la gestión de sus contenidos. Además, acumulan el número de bloques pedidos a memoria principal, el número de bloques expulsados, y el número de bloques escritos en memoria principal.

2. Conexión con el generador de traza

Una vez definidas las funciones que modelan la memoria cache, en este apartado las usaremos desde el simulador del procesador que hemos creado en prácticas anteriores. Como primer paso, solo realizaremos una simulación de cache a nivel de acierto/fallo, sin modificar el comportamiento temporal del procesador.

Para ello, en la etapa de búsqueda detectaremos las instrucciones load y store, e insertaremos las llamadas a leer() y escribir() correspondientes. Además, contabilizaremos el número de aciertos y fallos que se producen.

3. Estudiar comportamiento con distintos patrones

Los programas matrizc y matrizf, que ya conocéis, acceden a memoria usando patrones con distinta localidad espacial. En este apartado ejecutaremos ambos con nuestro simulador de cache y analizaremos las tasas de fallo obtenidas.

4. (Optativo) Modificar simulador temporal del procesador

Supondremos que disponemos de un predictor de acierto/fallo ideal. Usaremos este predictor en la ventana de lanzamiento para asignar latencia de acierto o latencia de fallo a cada instrucción load/store. Implementaremos nuestro predictor ideal accediendo a cache para determinar si almacena o no el bloque requerido.

Si optáis por realizar este apartado terminaremos de definirlo en la clase de laboratorio.