Soluciones a ejercicios de jerarquía de memoria

J. Daniel García Sánchez (coordinador)
David Expósito Singh
Javier García Blas
Óscar Pérez Alonso
J. Manuel Pérez Lobato

Arquitectura de Computadores
Grupo ARCOS
Departamento de Informática
Universidad Carlos III de Madrid

1. Ejercicios de examen

Ejercicio 1 *Examen de junio de 2015.*

Suponga que dispone de un computador con 1 ciclo de reloj por instrucción (CPI) cuando todos los accesos a memoria están en la cache. Los únicos accesos a datos son load y store, y suman el 25 % del total de instrucciones. La penalización por fallo es de 50 ciclos de reloj y la tasa de fallo del 5 %.

Determine el speedup que se obtiene cuando no hay ningún fallo de caché con respecto al caso en que se producen fallos de caché. Asuma que todas las instrucciones se encuentran en memoria caché.

Solución 1

Sea:

- c_p: Ciclos de procesador.
- c_w: Ciclos de espera.
- T: Periodo del reloj.
- IC: Número de instrucciones o *Instruction Count*.
- CPI: Ciclos por instrucción.
- a_i: Número de accesos a instrucción.
- a_d: Número de accesos a datos.
- m: Tasa de fallos.
- p_m: Penalización por fallo.

En el caso sin fallos:

$$T_{cpu} = (c_p + c_w) \cdot T = (IC \cdot CPI + 0) \cdot T = IC \cdot 1 \cdot T = IC \cdot T$$

Al introducir fallos:

$$c_w = IC \cdot (a_i + a_d) \cdot m \cdot p_m = IC(1 + 1 \cdot 0{,}25) \cdot 0{,}05 \cdot 50 = IC \cdot 1{,}25 \cdot 0{,}05 \cdot 50 = 3{,}125$$

Y por tanto el tiempo de cpu queda:

$$T_{cpu} = (IC \cdot 1 + IC \cdot 3{,}125) \cdot T = 4{,}125 \cdot IC \cdot T$$

Por tanto el speedup será:

$$S = \frac{4{,}125 \cdot IC \cdot T}{IC \cdot T} = 4{,}125$$

Ejercicio 2 *Examen de octubre de 2014.*

Sea la siguiente definición de variables globales:

```
const unsigned int max = 1024 * 1024;
double x[max];
double y[max];
double z[max];
double vx[max];
double vy[max];
double vz[max];
```

Y la siguiente función:

```
void actualiza_posiciones(double dt) {
  for (unsinged int i=0; i<max; ++i) {
    x[i] = vx[i] * dt + x[i];
    y[i] = vy[i] * dt + y[i];
    z[i] = vz[i] * dt + z[i];
  }
}
```

Suponga que se dispone de una caché L1 asociativa por conjuntos de 4 vías con un tamaño de 32 KB y un tamaño de línea de 64 bytes. La caché L2 es asociativa por conjuntos de 8 vías con un tamaño de 1 MB y un tamaño de línea de 64 bytes. La política de remplazo es LRU. Todas las cachés se encuentran inicialmente vacías.

Los arrays se almacenan de forma consecutiva en memoria y el primero ellos se encuentra en una dirección múltiplo de 1024. Asuma que el argumento de función **dt** y las variable índice **i** se encuentran asignadas a registros.

1. Determine la tasa de aciertos de las caché L1 y L2 para la ejecución de la función **actualiza_posiciones()**. ¿Cuál será la tasa global de aciertos?
2. Modifique el código aplicando la optimización de fusión de arrays.
3. Repita los cálculos del primer apartado para el código resultante del apartado segundo.
4. Si se asume que un acierto en la caché L1 requiere 4 ciclos, y en la caché L2 requiere 14 ciclos y la penalización por traer un bloque de memoria principal a la caché de nivel 2 es de 80 ciclos ¿cuál será el tiempo medio de acceso en cada uno de los dos casos?

Solución 2

Apartado 1 El patrón de accesos del bucle es:

vx[i], x[i], x[i], vy[i], y[i], y[i], vz[i], z[i], z[i], ...

Cada uno de los arrays tiene 2^{20} posiciones de 8 bytes cada una. Lo que da un total de 8 MB por array. Cada línea de la caché tiene 64 bytes, lo que permite alojar 8 elementos del array.

La caché L1 tiene 4 vías, por lo que cada vía es de 8KB (2^{13} bytes), lo que da para $\frac{2^{13}}{2^6} = 2^7$ conjuntos.

Como cada array tiene 2^{23} bytes (que es múltiplo del tamaño de vía), cada posición **i** de todos los arrays se corresponde con el mismo conjunto de la caché. La secuencia de aciertos y fallos en la caché será:

- **VX[0]** → Fallo. Selección del conjunto 0.
- **X[0]** → Fallo. Selección del conjunto 1.
- **X[0]** → Acierto.
- **VY[0]** → Fallo. Selección del conjunto 2.
- **Y[0]** → Fallo. Selección del conjunto 3.
- **Y[0]** → Acierto.
- **VZ[0]** → Fallo. Selección del conjunto 0. Se expulsa **vx**.
- **Z[0]** → Fallo. Selección del conjunto 1. Se expulsa **x**.
- **Z[0]** → Acierto.

Al pasar a la posición 1, se repite la misma sucesión de fallos y aciertos. Por lo que en total se tienen:

$$h_{L1} = \frac{3}{9} = \frac{1}{3}$$

Para la caché de nivel 2, se producen solamente accesos para los casos en que hay fallo en la caché L1. Para la posición 0, se producen 6 fallos, pero para las posiciones de la 1 a la 7 se producen 6 aciertos por cada posición.

$$h(L2) = \frac{42}{48} = \frac{7}{8}$$

La tasa global de aciertos h es:

$$H = 1 - M$$

Donde:

$$M = m_{h1} \cdot m_{h2} = \frac{2}{3} \cdot \frac{1}{8} = \frac{1}{12}$$

$$H = \frac{11}{12}$$

Apartado 2 Aplicando la optimización requerida el código queda:

```
const unsigned int max = 1024 * 1024;
struct part {
  double x, y, z, vx, vy, vz;
};
part vec[max];

void actualiza_posiciones(double dt) {
  for (unsinged int i=0; i<max; ++i) {
    vec[i].x = vec[i].vx * dt + vec[i].x;
    vec[i].y = vec[i].vy * dt + vec[i].y;
    vec[i].z = vec[i].vz * dt + vec[i].z;
  }
}
```

Apartado 3 El patrón de accesos en este caso es:

```
vec[i].vx, vec[i].x, vec[i].x, vec[i].vy, vec[i].y, vec[i].y, vec[i]. vz, vec[i].z, vec[i]. z, ...
```

Como cada posición del array ocupa 6 posiciones y una entrada de la caché ocupa 8 valores, en 3 líneas de la caché cabrán 4 posiciones completas del array.

- La posición 0 produce 1 fallo y 8 aciertos.
- La posición 1 produce 2 aciertos, 1 fallo y 6 aciertos.
- La posición 2 produce 4 aciertos, 1 fallo y 4 aciertos.
- La posición 3 produce 9 aciertos.

Este patrón se repite a lo largo de todo el array. Por tanto:

$$h_{L1} = \frac{33}{36} = \frac{11}{12}$$

Para la caché L2 se tiene ahora que todos los fallos en la caché de nivel 1, son también fallos en la caché de nivel 2. Por tanto:

$$h_{L2} = 0$$

Y la tasa global:

$$H = 1 - M$$

$$M = m_{h1} \cdot m_{h2} = \frac{1}{12}$$

$$H = \frac{11}{12}$$

Apartado 4 Para el caso original se tiene

$$T = 4 + (\frac{2}{3}) \cdot (14 + (\frac{1}{8}) \cdot 80) = 4 + (\frac{2}{3}) \cdot 24 = 4 + 16 = 20$$

Para el segundo caso se tiene

$$T = 4 + (\frac{1}{12}) \cdot (14 + 1 \cdot 80) = 4 + (\frac{1}{12}) \cdot 94 = 4 + 7{,}83 = 11{,}83$$

Ejercicio 3 *Examen de octubre de 2014.*

Sea la siguiente definición de variables globales:

```
const unsigned int max = 1024 * 1024;
double x[max];
double y[max];
double z[max];
double vx[max];
double vy[max];
double vz[max];
```

Y la siguiente función:

```
void actualiza_posiciones(double dt) {
  for (unsinged int i=0; i<max; ++i) {
    x[i] = vx[i] * dt + x[i];
  }
  for (unsinged int i=0; i<max; ++i) {
    y[i] = vy[i] * dt + y[i];
  }
  for (unsinged int i=0; i<max; ++i) {
    z[i] = vz[i] * dt + z[i];
  }
}
```

Suponga que se se dispone de una caché L1 asociativa por conjuntos de 4 vías con un tamaño de 32 KB y un tamaño de línea de 64 bytes. La caché L2 es asociativa por conjuntos de 8 vías con un tamaño de 1 MB y un tamaño de línea de 64 bytes. La política de remplazo es LRU. Todas las cachés se encuentran inicialmente vacías.

Los arrays se almacenan de forma consecutiva en memoria y el primero ellos se encuentra en una dirección múltiplo de 1024. Asuma que el argumento de función **dt** y las variable índice **i** se encuentran asignadas a registros.

1. Determine la tasa de aciertos de las caché L1 y L2 para la ejecución de la función **actualiza_posiciones()**. ¿Cuál será la tasa global de aciertos?

2. Modifique el código aplicando la optimización de fusión de bucles.

3. Repita los cálculos del primer apartado para el código resultante del apartado segundo.

4. Si se asume que un acierto en la caché L1 requiere 4 ciclos, y en la caché L2 requiere 16 ciclos y la penalización por traer un bloque de memoria principal a la caché de nivel 2 es de 80 ciclos ¿cuál será el tiempo medio de acceso en cada uno de los dos casos?

Solución 3

Apartado 1 Se ejecutan tres bucles consecutivamente.

El primer bucle tiene un patrón de accesos:

```
vx[i], x[i], x[i]
```

Los otros dos bucles tienen patrones de acceso similares.

Cada uno de los arrays tiene 2^{20} posiciones de 8 bytes cada una. Lo que da un total de 8 MB por array. Cada línea de la caché tiene 64 bytes, lo que permite alojar 8 elementos del array.

La caché L1 tiene 4 vías, por lo que cada vía es de 8KB (2^{13} bytes), lo que da para $\frac{2^{13}}{2^6} = 2^7$ conjuntos.

Como cada array tiene 2^{23} bytes (que es múltiplo del tamaño de vía), cada posición **i** de todos los arrays se corresponde con el mismo conjunto de la caché. La secuencia de aciertos y fallos en la caché será:

- **VX[0]** → Fallo. Selección del conjunto 0.
- **X[0]** → Fallo. Selección del conjunto 1.
- **X[0]** → Acierto.

Los siguientes 7 elementos de los vectores provocarán cada uno 3 aciertos. Lo mismo ocurrirá con los otros dos bucles.

Por tanto la tasa de aciertos para la cachés de nivel 1 será:

$$h_{L1} = \frac{22}{24} = \frac{11}{12}$$

A la caché de nivel 2, solamente irán los fallos de la caché L1. En este caso todos los casos generarán fallo.

$$h_{L2} = 0$$

La tasa global de aciertos h es:

$$H = 1 - M$$

Donde:

$$M = m_{L1} \cdot m_{L2} = \frac{1}{12}$$

$$H = \frac{11}{12}$$

Apartado 2 Aplicando la optimización requerida el código queda:

```
void actualiza_posiciones(double dt) {
  for (unsinged int i=0; i<max; ++i) {
    x[i] = vx[i] * dt + x[i];
    y[i] = vy[i] * dt + y[i];
    z[i] = vz[i] * dt + z[i];
  }
}
```

Apartado 3 El patrón de accesos del bucle es:

`vx[i], x[i], x[i], vy[i], y[i], y[i], vz[i], z[i], z[i], ...`

Cada uno de los arrays tiene 2^{20} posiciones de 8 bytes cada una. Lo que da un total de 8 MB por array. Cada línea de la caché tiene 64 bytes, lo que permite alojar 8 elementos del array.

La caché L1 tiene 4 vías, por lo que cada vía es de 8KB (2^{13} bytes), lo que da para $\frac{2^{13}}{2^6} = 2^7$ conjuntos.

Como cada array tiene 2^{23} bytes cada posición i de todos los arrays se corresponde con el mismo conjunto de la caché. La secuencia de aciertos y fallos en la caché será:

- **VX[0]** → Fallo. Selección del conjunto 0.
- **X[0]** → Fallo. Selección del conjunto 1.
- **X[0]** → Acierto.
- **VY[0]** → Fallo. Selección del conjunto 2.
- **Y[0]** → Fallo. Selección del conjunto 3.
- **Y[0]** → Acierto.
- **VZ[0]** → Fallo. Selección del conjunto 0. Se expulsa **vx**.
- **Z[0]** → Fallo. Selección del conjunto 1. Se expulsa **x**.
- **Z[0]** → Acierto.

Al pasar a la posición 1, se repite la misma sucesión de fallos y aciertos. Por lo que en total se tienen:

$$h_{L1} = \frac{3}{9} = \frac{1}{3}$$

Para la caché de nivel 2, se producen solamente accesos para los casos en que hay fallo en la caché L1. Para la posición 0, se producen 6 fallos, pero para las posiciones de la 1 a la 7 se producen 6 aciertos por cada posición.

$$h_{L2} = \frac{42}{48} = \frac{7}{8}$$

La tasa global de aciertos h es:

$$H = 1 - M$$

Donde:

$$M = m_{L1} \cdot m_{L2} = \frac{2}{3} \cdot \frac{1}{8} = \frac{1}{12}$$

$$H = \frac{11}{12}$$

Apartado 4 En el caso original, el tiempo medio de acceso es:

$$T = 4 + \frac{1}{12} \cdot (16 + 1 \cdot 80) = 4 + \frac{96}{12} = 4 + 8 = 12$$

En el caso modificado, el tiempo medio de acceso es:

$$T = 4 + \frac{2}{3} \cdot (16 + 80 \cdot \frac{1}{8} = 4 + \frac{2}{3} \cdot 26 = 4 + 17{,}33 = 21{,}33$$

Ejercicio 4 *Examen de octubre de 2013.*

Dado el siguiente fragmento de código:

```
struct particula {
  double x, y;
  double ax, ay;
  double vx, vy;
  double masa;
  double carga;
};

void mueve(particula p[], int n, double t) {
  for (int i=0; i<n; ++i) {
    p[i].x += p[i].vx * t + 0.5 *p[i].ax * t * t;
  }
  for (int i=0; i<n; ++i) {
    p[i].y += p[i].vy * t + 0.5 * p[i].ay * t * t;
  }
}
```

La función **mueve()** se ejecuta para un array de 1000 partículas, en un sistema con una memoria caché de L1 de datos de 32 KB asociativa por conjuntos de 4 vías y tamaño de bloque de 64 bytes. La caché se encuentra inicialmente vacía.

Asuma que el array p[] está alineado a una dirección múltiplo de 64. Asuma que los argumentos **n** y **t**, así como la variable de índice **i** se encuentran asignadas a registros.

Se pide determinar:

1. El número de fallos de cache.
2. La tasa media de fallos.
3. Proponga un código alternativo usando la técnica de fusión de bucles. ¿Cuál sería la nueva tasa de fallos?
4. Proponga un código alternativo reorganizando el código en múltiples arrays paralelos y sin fusión de bucles ¿Cuál sería la nueva tasa de fallos?
5. ¿Cree que es conveniente aplicar la fusión de bucles en el tercer apartado? Justifique su respuesta.

Solución 4

Apartado 1 En cada bloque de caché se pueden almacenar 8 valores double. Por otra parte cada partícula requiere también 64 bytes. Además el array completo tiene 1000 posiciones por lo que no cabe completamente en la caché.

Durante el primer bucle cada iteración produce un fallo al acceder para leer a **p[i].x** y 3 aciertos, al leer **p[i].ax** y **p[i].vx**, y al escribir **p[i].x**.

Durante el segundo bucle ocurre exactamente lo mismo.

Por tanto, se producen en total 2000 falllos.

Apartado 2 La tasa de fallos es:

$$m = \frac{2000}{8000} = 0{,}25$$

Apartado 3 Si se aplica la técnica de fusión de bucles se tiene:

```
void mueve(particula p[], int n, double t) {
  for (int i=0; i<n; ++i) {
    p[i].x += p[i].vx * t + 0.5 *p[i].ax * t * t;
    p[i].y += p[i].vy * t + 0.5 * p[i].ay * t * t;
  }
}
```

En cada iteración del bucle, el acceso a **p[i].x** genera un fallo, pero el resto de accesos genera aciertos, llegándose a una tasa de fallos de:

$$m = \frac{1000}{8000} = 0{,}125$$

Apartado 4 Se podría tener el siguiente código:

```
struct particulas {
  double x[1000], y[1000];
  double ax[1000], ay[1000];
  double vx[1000], vy[1000];
  double masa[1000];
  double carga[1000];
};

void mueve(particulas * p, int n, double t) {
  for (int i=0; i<n; ++i) {
    x[i] += vx[i] * t + 0.5 *ax[i] * t * t;
  }
  for (int i=0; i<n; ++i) {
    y[i] += vy[i] * t + 0.5 * ay[i] * t * t;
  }
}
```

Ahora se tienen arrays independientes. En cada bloque de caché caben 8 posiciones de un array.

En el primer bucle, la primera iteración produce 3 fallos y un acierto. Las siguientes 7 iteraciones producen 4 aciertos por iteración (28 aciertos en total). Este patrón se repite 125 veces en cada bucle.

Tasa de aciertos:

$$h = 3 \cdot 125 \cdot \frac{2}{8000} = \frac{750}{8000} = 0{,}094$$

Apartado 5 No es conveniente porque el número de arrays paralelos es mayor que el número de vías.

Ejercicio 5 *Examen de octubre de 2013.*

Dado el siguiente fragmento de código:

```
for (i=0; i<64; i++)
  for (j=0; j<1024; j=j+2 )
    v[i][j] = v[i][j] * v[i][j+1] + b[i]
```

Se ejecuta en una arquitectura con una cache de tamaño 256 KB, con tamaño de bloque 64 Bytes y de palabra de 8 Bytes en el que la memoria caché es totalmente asociativa y utiliza un algoritmo de reemplazo LRU. Siendo **v** y **b** matrices de números reales de 8 byte almacenadas por filas (estilo C) y tamaños 64×1024 (**v**), 64 (**b**). Se supondrá que las variables índice se almacenan en registros y que al empezar el código la caché está vacía.

Se pide determinar:

Acceso a **v**	Fallo caché	Acceso a **b**	Fallo caché
0,0	SI	0	SI
0,2	No. Porque está en la misma línea de cache que **v[0,0]**	0	No, Porque está en la misma línea de cache que **b[0]**
...		0	No
0,7	NO	0	No
0,8	SI	0	No
0,10	NO	0	No
... 0,16	SI	0	No
... 0,1022	NO	0	No
1,0	SI	1	NO
1,1	No porque está en la misma línea de cache que **v[1,0]**	1	NO
... 2,0	SI	2	NO
... 8,0	SI	8	SI
... 31,0	SI	31	NO

Cuadro 1: Secuencia de accesos del ejercicio 5.

1. El número de fallos de cache.
2. La tasa media de fallos.
3. Razone si se produce alguna vez reemplazo de una línea de cache precargada. No es necesario identificar qué reemplazos hay sino razonar la existencia o no de reemplazos.
4. Razone si la técnica de optimización de intercambio de bucles mejoraría la tasa media de fallos de la caché (no es necesario calcularla).

Solución 5

Apartado 1 Cache de 256 KB $\rightarrow 2^{15}$ elementos $= 2^{12}$ bloques

Bloque de 64B = 8 elementos

La Tabla 1 muestra la secuencia de accesos.

Se producen fallos de cache al acceder a **v[0,0]**, **v[0,8]**, **v[0,16]**, ... **v[1,0]**, **v[1,8]**, **v[0,16]**, ... **[2,0]**, ... **v[31,0]**, ... y hay aciertos en todas las demás. Como la matriz tiene 64×1024 casillas y sólo hay error en una de cada 8 el total de fallos es $\frac{64 \cdot 1024}{8} = 8192$ fallos

En **b** se producen fallos en las posiciones 1, 8, 16, 24, 32, 40, 48 y 56. En total se producen 8 fallos.

El número total fallos es $8192 + 8 = 8200$ fallos.

Apartado 2 Accesos totales.

En cada instrucción hay 4 accesos y la instrucción de cálculo se realiza $\frac{64 \cdot 1024}{2}$ veces. Por tanto, tenemos $\frac{4 \cdot 64 \cdot 1024}{2} = 131072$ accesos u la tasa de fallos es:

$$m = \frac{8200}{131072} = 0{,}0625 \Rightarrow 6{,}25\,\%$$

Apartado 3 En la cache caben 256 KB = 2^{15} elementos = 2^{12} bloques y la matriz v tiene $64 \cdot 1024 = 2^{16}$ elementos = 2^{13} bloques.

Luego la matriz **v** no cabe en la caché. Se producirá un reemplazo de la línea que contiene la posición **v[0,0]** cuando lleguemos, aproximadamente, a la mitad de la matriz **v**.

Si no existiera la matriz **b** el reemplazo se produciría al llegar a la mitad de la matriz (pues en caché sólo cabe la mitad de la matriz), es decir en la posición **v[32,0]**.

Teniendo en cuenta que la matriz **b** también ocupa cache y cuando llegamos a la posición **v[32,0]** hemos ocupado 4 líneas de caché con la matriz **b** (posiciones de la **b[0]** a la **b[32]**) podemos concluir que el reemplazo se producirá 4 líneas de cache antes de llegar a la **v[32,0]**. Esto es en la posición $\mathbf{v}[31, 8 \cdot \frac{1024}{8} - 4] = \mathbf{v[31,992]}$. Este dato no se pide explícitamente en el ejercicio.

De igual forma se producirá un reemplazo de la línea de caché de **b[0]** en, aproximadamente, el mismo momento.

También habrá reemplazos sucesivos de **v[0,8]**, **v[0,16]**, ...según se va necesitando sitio para las posiciones de **v** que están más allá de la mitad del recorrido.

Apartado 4 El intercambio de bucles disminuiría la localidad espacial en los accesos a **v**, aumentando la tasa de fallos.

Ejercicio 6 *Examen de octubre de 2013.*

Dado el siguiente fragmento de código:

```
int a[100];
int b[100];
for (i=0;i<100;i++)
  a[i]=a[i]+5;
for (i=0;i<100;i++){
  if (i>90)
    c=c+1;
  else
    b[i]=a[i]*3;
}
```

Se pide:

1. Describir las optimizaciones de caché del compilador que permitan mejorar el tiempo de acceso a la memoria para este código y reescríbalo de acuerdo a dichas optimizaciones.

2. Considere un computador con una memoria caché cuya capacidad de línea es 32 bytes. Inicialmente la caché está vacía. ¿Cuál es la tasa de fallos de la caché en la ejecución del programa optimizado obtenido en el apartado anterior? Considere sólo fallos forzosos de los vectores **a** y **b** (no existen fallos de capacidad o conflicto).

3. Considerar ahora un computador con una memoria multinivel L1 y L2 ambas unificadas. El tiempo de ciclo es de 1ns y CPI=1.3. Para la caché de nivel 1 se asume una tasa de fallos del 10 % y una penalización de 10 ns. Para la caché de nivel 2 la tasa de aciertos es del 95 % y la penalización de L2 es 80ns. Se asume un tiempo de acceso a la caché L1 de 1ns. Se pide calcular el tiempo de ejecución de un programa con CI instrucciones donde el 50 % de las instrucciones son de Lectura/Escritura.

Solución 6

Apartado 1 Las optimizaciones de caché del compilador que se pueden aplicar son las siguientes:

- **Fusión de arrays**: los arrays **a** y **b** al ser del mismo tamaño se pueden fusionar en uno solo de tal manera que se defineun espacio consecutivo de memoria para almacenar los dos elementos del array, mejorando la localidad espacial y reduciendo así los fallos de caché.
- **Fusión de bucles**: se pueden fusionar los bucles en la línea 1 y 3 que iteran sobre los arrays **a** y **b** respectivamente en un único bucle, de manera que se aprovechen los accesos al vector a para realizar todas las operaciones sobre **a**, sin que tengamos un nuevo fallo de caché.
- **Linearización**: el compilador puede cambiar el sentido de la condición si detecta que la estrategia de predicción de salto generará muchos errores dado que no coincide con el resultado de la condición.

Una vez aplicadas las optimizaciones de caché, un posible código resultante sería el siguiente:

```
struct mi_array {
  int a;
  int b;
};

struct mi_array array[100];
for (i=0;i<100;i++) {
  array[i].a= array[i].a+5;
  if (i<=90)
    array[i].b=array[i].a*3;
  else
    c=c+1;
}
```

Apartado 2 Dado que el tamaño de línea de caché es 32 B en cada línea de caché caben $\frac{32}{4} = 8$ datos de tipo **int**.

- Accesos **a**
  - **array[0].a** → fallo de caché, transfiere 32 B. **array[0].a, array[0].b, array[1].a, array[1].b, array[2].a, array[2].b, array[3].a, array[3].b**
  - **array[1].a** ... **array[3].a** → aciertos de caché.
  - **array[4].a** → fallo de caché, transfiere 32B. **array[4].a,array[4].b,array[5].a,array[5].b,array[6].a,ar array[7].a, array[7].b**
  - **array[5].a** ... **array[7].a** → aciertos de caché.
  - ...
  - **array[96].a** → fallo de caché, transfiere 32B. **array[96].a, array[96].b, array[97].a, array[97].b, array[98].a, array[98].b, array[99].a, array[99].b**
- Accesos **b**
  - **array[0].b** → No hay fallo de caché puesto que el acceso se realiza inmediatamente después de haber tenido el fallo de lectura de **array[0]**.
  - **array[1].b** → No hay fallo de caché puesto que el acceso se realiza inmediatamente después de haber tenido el fallo de lectura de **array[0]**.
  - ...

- **array[99].b** → No hay fallo de caché puesto que el acceso se realiza inmediatamente después de haber tenido el fallo de lectura de **array[0]**.

Ningún fallo de caché debido a **b**. Se tienen 90 accesos a **b** y ningún fallo.

El número de accesos totales es $100 + 100 + 90 + 90 = 380$ accesos a memoria. 1 de cada 4 accesos al vector **a** es un fallo de caché. Dado que se tiene 100 accesos, 25 son fallos.

Tasa de fallos.

$$m = \frac{25}{380} = 6{,}57\,\%$$

Apartado 3

$$t = t_{cpu} + t_{mem}$$

$$t_{cpu} = CI \cdot CPI \cdot T = CI \cdot 1{,}3 \cdot 1 = 1{,}3 \cdot CI$$

$$t_{mem} = t_{fetch} + t_{datos}$$

$$t_{fetch} = t_{acceso} + t_{fallosL1} \cdot (penalizacion_{L1} + (t_{fallosL2} \cdot penalizacion_{L2})) = 1 + (0{,}1 \cdot (10 + (0{,}05 \cdot 80)) = 2{,}4$$

El número medio de ciclos debido al fetch de las instrucciones es

$$t_{fetch} = CI \cdot 2{,}4 \cdot T = 2{,}4 \cdot CI$$

$$t_{datos} = t_{acceso} + (t_{fallosL1} \cdot (penalizacion_{L1} + (t_{fallosL2} \cdot penalizacion_{L2})) = 1 + (0{,}1 \cdot (10 + (0{,}05 \cdot 80)) = 2{,}$$

$$t_{datos} = CI \cdot \frac{num\_ref\_memoria}{IC} \cdot 2{,}4 \cdot T = 2{,}4 \cdot CI \cdot 0{,}5 \cdot 1 = 1{,}2 \cdot CI$$

$$t_{mem} = 2{,}4 \cdot CI + \cdot 1{,}2 \cdot CI = 3{,}6 \cdot CI$$

$$Total = 1{,}3 \cdot CI + 3{,}6 \cdot CI = 4{,}9 \cdot CI$$

Ejercicio 7 *Examen de junio de 2011.*

Sea el siguiente fragmento de código:

```
double a[256][256], b[256][256], c[256][256], d[256][256];
//...
for (int i=0;i<256;++i)
  for (int j=0;j<256;++j) {
    a[i][j] = b[i][j] + c[i][j]
  }
}
for (int i=0;i<256;++i)
  for (int j=0;j<256;++j) {
    d[i][j] = b[i][j] - c[i][j]
  }
}
```

Se desea ejecutar este código en un computador que tiene una caché de nivel 1 totalmente asociativa de 16KB y política de sustitución LRU con tamaño de línea de 64 bytes. Los fallos de caché de nivel 1 requieren 16 ciclos de reloj. Por otra parte la caché de nivel 2 siempre genera aciertos en este código.

Asuma que los fallos de escritura en la caché de nivel 1 se envían directamente un búfer de escritura y no generan ningún ciclo de espera.

Se pide

1. Determine la tasa de aciertos del segmento de código, asumiendo que las variables **i** y **j** se asignan a registros del procesador y que las matrices **a**, **b**, **c** y **d** se encuentran totalmente en la caché de nivel 2.
2. Determine el tiempo acceso a memoria asumiendo que los accesos a la caché de nivel 1 requieren un ciclo de reloj.
3. Proponga una transformación de código de las que puede generar un compilador para mejorar la tasa de aciertos, mostrando el código resultante en lenguaje C.
4. Determine la nueva tasa de aciertos y el tiempo medio de acceso a memoria resultantes.

Solución 7

Apartado 1 Como el tamaño de línea es de 64 bytes y todos los arrays contienen valores de tipo **double** cada línea de la memoria caché puede almacenar 8 valores. Como la memoria L1 tiene un tamaño de 16 KB, puede almacenar $\frac{2^{14}}{2^6} = 2^8$ líneas.

Cada array almacena $2^8 \dot{2}^8 = 2^{16}$ valores, lo que requiere $\frac{2^{16}}{2^3} = 2^{13}$ entradas de la memoria caché. En el primer bucle el patrón de accesos es:

- **b[0][0]**, **c[0][0]**, **a[0][0]**
- **b[0][1]**, **c[0][1]**, **a[0][1]**
- ...
- **b[0][7]**, **c[0][7]**, **a[0][7]**
- **b[0][8]**, **c[0][8]**, **a[0][8]**
- ...
- **b[0][255]**, **c[0][255]**, **a[0][255]**
- **b[1][0]**, **c[1][0]**, **a[1][0]**
- ...

Por tanto por cada 8 iteraciones del bucle interior se producen:

- 16 lecturas (8 para **b** y 8 para **c**). De estas 2 son fallos y las otras 14 son aciertos.
- 8 escrituras en **a**. Sin embargo al enviarse las escrituras al búfer de escrituras, éstas no generan fallos.

Cuando se inicia el segundo bucle, las entradas de la caché conteniendo los valores de los arrays **b** y **c** han sido expulsadas de la caché, por lo que la proporciones de fallos y aciertos es la misma.

En el caso de las escrituras, todas la escrituras generan fallo de caché. Por tanto:

$$h_{escr} = 0$$

En el caso de las escrituras, la tasa de aciertos es:

$$h_{lect} = \frac{14}{16} = \frac{7}{8}$$

Apartado 2 Para los accesos de escritura, todos los accesos son un fallo de escritura, y se tratan en el búfer de escritura, requiriendo un ciclo de reloj. En total se realizan $2^8 \cdot 2^8$ accesos en cada uno de los dos bucles, lo que da un total de $2 \cdot 2^{16} = 2^{17}$ accesos de escritura.

Para los accesos de lectura, el tiempo medio de acceso viene dado por:

$$t_{lect} = t_a + (1 - h_{lect}) \cdot t_f$$

El tiempo de acceso t_a es 1 ciclo. La penalización por fallo de lectura es de 16 ciclos. Por tanto se tiene que el tiempo medio de lectura es:

$$t_{lect} = 1 + \frac{1}{8} \cdot 16 = 3$$

Apartado 3 Se puede aplicar la técnica de fusión de bucles:

```
for (int i=0;i<256;i++) {
  for (int j=0;j<256;j++) {
    a[i][j] = b[i][j] + c[i][j];
    d[i][j] = b[i][j] - c[i][j];
  }
}
```

Apartado 4 En este caso los segundos accesos a **b** y **c** siempre generan acierto.

Ahora por cada 8 iteraciones del bucle interior se producen:

- 32 lecturas. De éstas, 30 son aciertos y dos son fallos de lectura.
- 16 escrituras.

Al igual que en el caso anterior la tasa de aciertos de escritura (h_{escr}) sigue siendo 0. En este caso, la tasa de aciertos de lectura pasa a ser:

$$h_{lect} = \frac{30}{32} = \frac{15}{16}$$

Para el caso de los accesos de lectura, usando la nueva tasa de aciertos, se tiene:

$$t_{lect} = 1 + \frac{1}{16} \cdot 16 = 2$$

Ejercicio 8 *Examen de mayo de 2011.*

Dada una arquitectura con dos niveles caché con las siguientes características:

Memoria	Tiempo de acceso (ns)	Tasa de aciertos
L1	2	0.8
L2	8	0.9
RAM	100	1

El ordenador ejecuta un programa que reside completamente en memoria (no hay accesos a disco). Se pide:

1. Asumiendo que el 100 % de los accesos a memoria son operaciones de escritura, calcular de forma justificada el tiempo medio de acceso a memoria para (a) una política de escritura inmediata (write-through) y (b) una política de post-escritura (write-back) en las memorias caché L1 y L2.

2. Considerando los dos niveles de memoria caché L1 y L2 como una única caché global se pide calcular el tiempo medio de acceso y la tasa de aciertos de esta caché global.

3. Dado el siguiente código:

```
for (i=0;i<1000;i=i+32){
  a[i]=a[i+8]+a[i+16];
}
```

   El tamaño de cada entrada de a es de 8 bytes y el de bloque de 64 bytes. El índice del bucle se almacena en un registro del procesador. Se pide comentar razonablemente el efecto en el rendimiento del empleo de una técnica de caché multibanco que emplea 4 bancos. ¿Qué repercusión tendría el empleo de esta técnica en el tiempo de cada acceso y en el ancho de banda de la caché para este código?

Solución 8

Apartado 1 En el caso de escritura inmediata, como el 100 % de los accesos son operaciones de escritura, todos ellos se realizan en memoria principal. Por este motivo, el tiempo de acceso será en todos los casos:

$$t = 2 + 8 + 100 = 110ns$$

En el caso de post escritura,si se produce un acierto caché no es necesario escribir en memoria. De este modo el tiempo de acceso será:

$$t = 2 + (1 - 0{,}8) \cdot (8 + (1 - 0{,}9) \cdot 100) = 2 + 0{,}2 \cdot (8 + 0{,}1 \cdot 100) = 2 + 0{,}2 \cdot 18 = 2 + 3{,}6 = 5{,}6ns$$

Apartado 2 El tiempo medio de acceso es:

$$t = 2 + (1 - 0{,}8) \cdot 8 = 2 + 0{,}2 \cdot 8 = 2 + 1{,}6 = 3{,}6ns$$

Y la tasa media de aciertos:

$$h = 1 - (1 - 0{,}8) \cdot (1 - 0{,}9) = 1 - 0{,}2 \cdot 0{,}1 = 1 - 0{,}02 = 0{,}98.$$

Apartado 3 En cada iteración del bucle se acceden a tres bloques distintos que están ubicados en posiciones consecutivas de memoria. Mediante el empleo de una memoria caché no multibanco, se producirían 3 fallos caché que tendrían que ser atendidos secuencialmente. En el caso de una caché multibanco, sería posible acceder de forma paralela a estos tres bloques.

El tiempo de acceso sería el mismo que antes, debido a que esta optimización no mejora el tiempo de acceso a la memoria caché.

El ancho de banda sería el agregado, ya que se estarían haciendo accesos en paralelo. En este caso, serían 3 accesos simultáneos por lo que el ancho de banda sería el triple.