Escuela de Ingeniería y Arquitectura
Universidad Zaragoza

Departamento de Informática e Ingeniería de Sistemas
Universidad Zaragoza

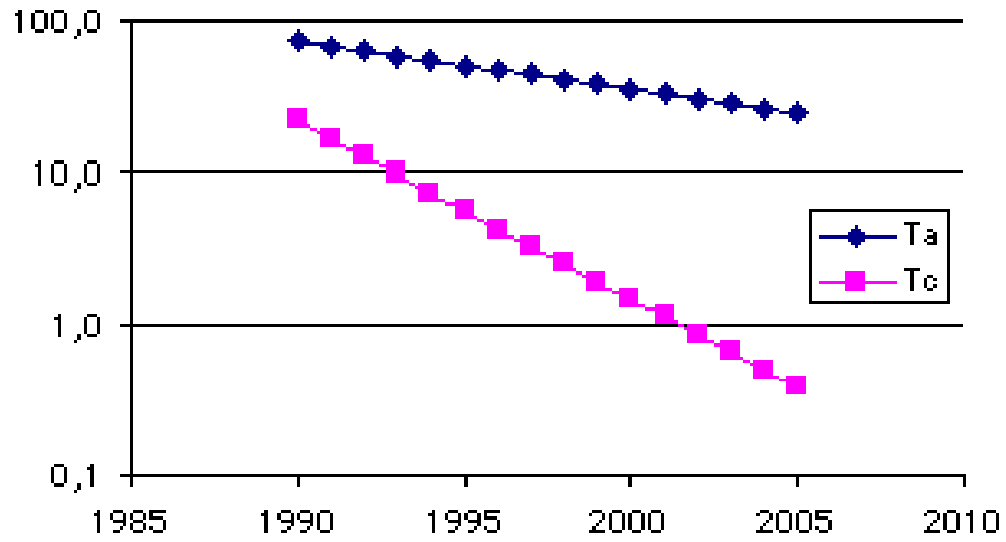
Tema 6 – Incremento de rendimiento

P. Ibáñez, J.L. Briz, V. Viñals, J. Alastruey, J. Resano
Arquitectura y Tecnología de Computadores
Departamento de Informática e Ingeniería de Sistemas

Guión del tema

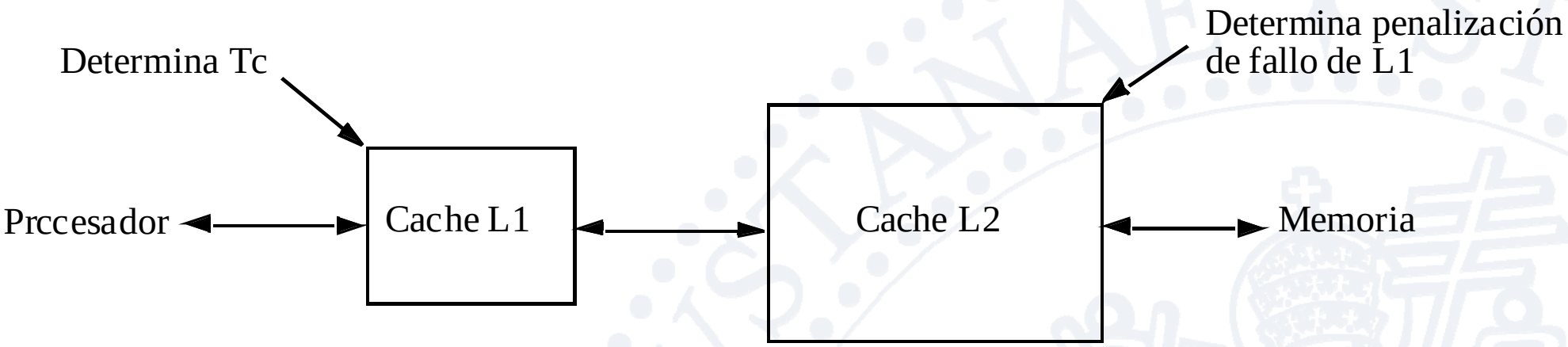
- El problema de la creciente penalización del fallo
- Organización multinivel
- Optimización de código por el compilador

Problema: creciente penalización del fallo



- Objetivos memoria cache
  - Servir al procesador a su ritmo
    - Diseñar cache pequeña y sencilla
  - Minimizar la tasa de fallos
    - Diseñar cache grande y compleja

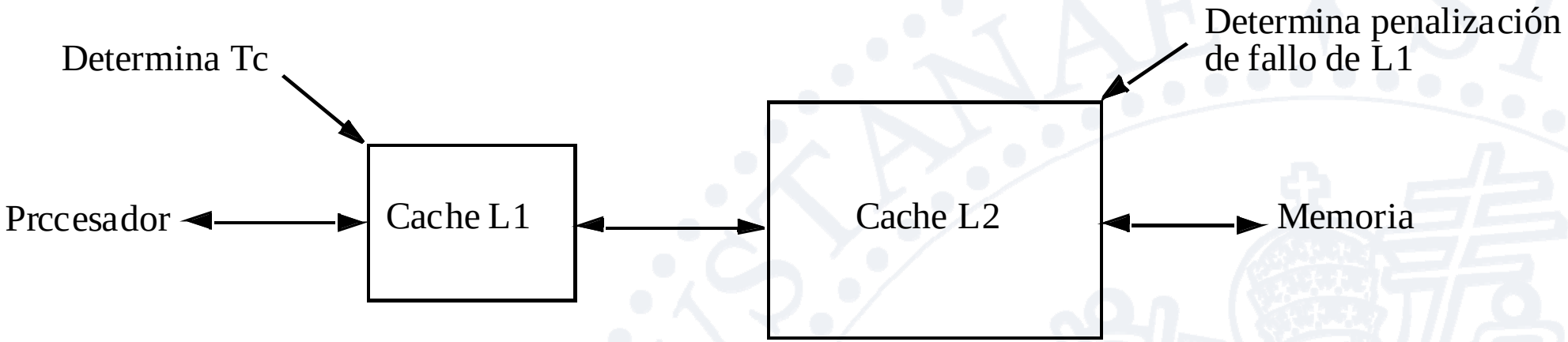
Organización multinivel



$$Tasa\ \ fallos\ L_1 = m_1 = \frac{fallos\ L_1}{accesos\ L_1} \qquad Tasa\ fallos\ L_2 = m_2 = \frac{fallos\ L_2}{accesos\ L_2}$$

$$Tasa\ \ fallos\ global\ = m_g = \frac{fallos\ L_2}{accesos\ L_1} = \frac{fallos\ L_2}{accesos\ L_2} \times \frac{fallos\ L1}{accesos\ L_1} = m_2 \times m_1$$

Organización multinivel



$$Tef = Th_{L1} + m_1 * P_{L1} = Th_{L1} + m_1 \times \left[Th_{L2} + m_2 \times P_{L2} \right]$$

- P_{L1}: penalización por fallo de L1
- Th_{L2}: tiempo para transferir un bloque de L2 a L1
- P_{L2}: tiempo para transferir un bloque de Mp a L2 y L1

Parámetros de diseño en L2

- Objetivo diseño L2: minimizar tasa de fallos
  - Su tiempo de acceso no es el factor más determinante en Ta

$$Tef = Th_{L1} + m_1 \times P_{L1} = Th_{L1} + m_1 \times \left[Th_{L2} + m_2 \times P_{L2} \right]$$

- Tamaño mucho mayor que L1
- Asociatividad alta
- Tamaño de bloque mayor que en L1
- ¿Política de escritura?

- Inclusión de contenidos

Ejercicio

- Se tienen dos jerarquías, una con un nivel de cache (L1) y otra con dos niveles (L1+L2)
  - Tasa de fallos en L1: m_1= 0,1 (10%)
  - Tasa local de fallos en L2: m_2= 0,5 (50%)
  - Acierto en L1: 1 ciclo
  - Acierto en L2: 6 ciclos
  - Penalización por acceso a Mp: 100 ciclos
- Calcular para las 2 jerarquías
  - Tasa de fallos global
  - Ciclos efectivos de acceso, y Tef si F = 2 GHz
- Calcular T_{ef} suponiendo m1, m2 y F igual, pero:
  - Ta(L1) = 0.5 ns; Ta(L2) = 2.8 ns; P_{Mp}= 49.55 ns

Optimización de código por el compilador

- Fusión de vectores

```
int val[SIZE], key[SIZE];
for (i=0; i<SIZE; i=i+20) {
   if (key[i] < x)
      acum = acum + val[i];
}
```

```
struct merge {
   int val;
   int key;
}
struct merge merged_array[SIZE];
for (i=0; i<SIZE; i=i+20) {
   if (merged_array[i].key < x)
      acum = acum + merged_array[i].val;
}
```

Optimización de código por el compilador

- Intercambio de bucles (*loop interchange*)

```
for (j=0; j<100; j++) {
   for (i=0; i<5000; i++)
      x[i][j] = 2*x[i][j];
}
```

```
for (i=0; i<5000; i++) {
   for (j=0; j<100; j++)
      x[i][j] = 2*x[i][j];
}
```

Optimización de código por el compilador

- Fusión de bucles (*loop fusion*)

```
for (i=0; i<N; i++) {
   for (j=0; j<N; j++)
      a[i][j]= 1/ b[i][j]* c[i][j];
}
for (i=0; i<N; i++) {
   for (j=0; j<N; j++)
      d[i][j]= a[i][j] + c[i][j];
}
```

```
for (i=0; i<N; i++) {
   for (j=0; j<N; j++) {
      a[i][j]= 1/ b[i][j]* c[i][j];
      d[i][j]= a[i][j] + c[i][j];
   }
}
```

Reduce sobrecarga bucles, aumenta localidad …

Optimización de código por el compilador

- Fisión de bucles (*loop fission*)

```
for (i = 0; i < n; i++)
   y[i] = y[i] + x[i] + x[i+m];
```

Bajo rendimiento si x[i] y x[i+m] coinciden en el mismo bloque de cache (ej: mapeo directo y m potencia de 2)

```
for (i = 0; i < n; i++)
   y[i] = y[i] + x[i];

for (i = 0; i < n; i++)
   y[i] = y[i] + x[i+m];
```

Optimización de código por el compilador

- Distribución de bucles (*loop distribution*)

```
for (i = 0; i < n; i++) {
   x[i] = y[i] + z[i] + w[i];     /* S1 */
   a[i+1] = (a[i-1] + a[i])/2.0; /* S2 */
   y[i] = z[i] - x[i];            /* S3 */
}
```

S1 y S3 pueden paralelizarse, S2 no.

→ El bucle **no** puede paralelizarse

```
/* B1 */
for (i = 0; i < n; i++) {
   x[i] = y[i] + z[i] + w[i];     /* S1 */
   y[i] = z[i] - x[i];            /* S3 */
}
/* B2 */
for (i = 0; i < n; i++) {         /* L2 */
   a[i+1] = (a[i-1] + a[i])/2.0; /* S2 */
}
```

B1 puede paralelizarse, B2 se ejecutará de forma secuencial