Optimizaciones avanzadas de memoria caché

Arquitectura de Computadores

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

Grupo ARCOS
Departamento de Informática
Universidad Carlos III de Madrid

¿Por qué aparece la memoria caché?

- Para vencer el muro de memoria (*memory wall*).
  - 1980 – 2010: Mejora de rendimiento de procesadores mayor (órdenes de magnitud) que de la memoria.
  - 2005 – . . . : Situación agravada por arquitecturas *multi-core*.

- Para reducir el tiempo de acceso a datos e instrucciones.
  - Aproximar tiempo de acceso a memoria al tiempo de acceso a la caché.
  - Ofrecer la ilusión de tamaño de la caché aproximando al tamaño de memoria principal.
  - Basado en el principio de localidad.

Tiempo medio de acceso a memoria

- 1 nivel.

$$t = t_h(L1) + m_{L1} \times t_p(L1)$$

- 2 niveles.

$$t = t_h(L1) + m_{L1} \times (t_h(L2) + m_{L2} \times t_p(L2))$$

- 3 niveles.

$$t = t_h(L1) + m_{L1} \times (t_h(L2) + m_{L2} \times (t_h(L3) + m_{L3} \times t_p(L3)))$$

- ...

Optimizaciones básicas

1. Aumentar el tamaño de bloque.
2. Aumentar el tamaño de caché.
3. Incrementar la asociatividad.
4. Introducir cachés multinivel.
5. Dar prioridad a fallos de lectura.
6. Evitar traducción de direcciones durante el indexado.

Optimizaciones avanzadas

- **Métricas a reducir**:
  - Tiempo de búsqueda (*hit time*).
  - Tasa de fallos (*miss rate*).
  - Penalización de fallos (*miss penalty*).
- **Métricas a aumentar**:
  - Ancho de banda de caché (*cache bandwidth*).
- **Observación**: Todas las optimizaciones avanzadas buscan mejorar alguna de estas métricas.

2 Optimizaciones avanzadas

- Cachés pequeñas y simples
- Predicción de vía
- Acceso segmentado a la caché
- Cachés no bloqueantes
- Cachés multi-banco
- Palabra crítica primero y reinicio temprano
- Mezclas en búfer de escritura
- Optimizaciones del compilador
- Lectura adelantada hardware

Cachés pequeñas

- Proceso de **búsqueda**:
  - Selección de línea utilizando índice.
  - Lectura de etiqueta de línea.
  - Comparación con etiqueta de dirección.
- El tiempo de búsqueda se **incrementa** con el tamaño de la caché.
- Una caché **más pequeña** permite:
  - Hardware de búsqueda más simple.
  - La caché cabe en el chip del procesador.
- **Una caché pequeña mejora el tiempo de búsqueda**.

Cachés simples

- Simplificación de la caché.
  - Uso de mecanismos de correspondencia lo más **sencillo** posibles.
  - **Correspondencia directa**:
    - Se permite **paralelizar** la comparación de la etiqueta con la transmisión del dato.

- **Observación**: La mayoría de los procesadores modernos se centran más en usar cachés pequeñas que en su simplificación.

Intel Core i7

- Caché L1 (1 por núcleo)
  - 32 KB instrucciones
  - 32 KB datos
  - Latencia: 3 ciclos
  - Asociativa 4(i), 8(d) vías.
- Caché L2 (1 por núcleo)
  - 256 KB
  - Latencia: 9 ciclos
  - Asociativa 8 vías.
- Caché L3 (compartida)
  - 8 MB
  - Latencia: 39 ciclos
  - Asociativa 16 vías.

2 Optimizaciones avanzadas

- Cachés pequeñas y simples
- Predicción de vía
- Acceso segmentado a la caché
- Cachés no bloqueantes
- Cachés multi-banco
- Palabra crítica primero y reinicio temprano
- Mezclas en búfer de escritura
- Optimizaciones del compilador
- Lectura adelantada hardware

Predicción de vía

- **Problema**:
  - **Correspondencia directa** → rápida pero muchos fallos.
  - **Asociativa por conjuntos** → menos fallos pero más conjuntos (más lenta).

- **Predicción de vía**
  - Se almacenan bits adicionales para predecir la vía que se seleccionará en el próximo acceso.
  - Acceso adelantado al bloque y comparación con una única etiqueta.
    - Si hay fallo se compara con las demás etiquetas.

2 Optimizaciones avanzadas

- Cachés pequeñas y simples
- Predicción de vía
- Acceso segmentado a la caché
- Cachés no bloqueantes
- Cachés multi-banco
- Palabra crítica primero y reinicio temprano
- Mezclas en búfer de escritura
- Optimizaciones del compilador
- Lectura adelantada hardware

Acceso segmentado a la caché

- **Objetivo**: Aumentar el ancho de banda de la caché.
- **Solución**: Segmentar el acceso a la caché en varios ciclos de reloj.
- **Efectos**:
  - Se puede reducir el ciclo de reloj.
  - Se puede iniciar un acceso en cada ciclo de reloj.
  - Se aumenta el ancho de banda de la caché.
  - Se aumenta la latencia.
- Efecto positivo en procesadores superescalares.

2 Optimizaciones avanzadas

- Cachés pequeñas y simples
- Predicción de vía
- Acceso segmentado a la caché
- Cachés no bloqueantes
- Cachés multi-banco
- Palabra crítica primero y reinicio temprano
- Mezclas en búfer de escritura
- Optimizaciones del compilador
- Lectura adelantada hardware

Cachés no bloqueantes

- **Problema**: Fallo de caché genera una parada (*stall*) hasta que se obtiene el bloque.
- **Solución**: Ejecución fuera de orden.
  - **Pero**: ¿Cómo se accede a la caché mientras se resuelve el fallo?
- **Acierto durante fallo**:
  - Permitir accesos con acierto mientras se espera.
  - Reduce la penalización del fallo.
- **Acierto durante fallos / Fallo durante fallo**:
  - Permitir fallos solapados.
  - Necesita memoria multi-canal.
  - Altamente complejo.

2 Optimizaciones avanzadas

- Cachés pequeñas y simples
- Predicción de vía
- Acceso segmentado a la caché
- Cachés no bloqueantes
- Cachés multi-banco
- Palabra crítica primero y reinicio temprano
- Mezclas en búfer de escritura
- Optimizaciones del compilador
- Lectura adelantada hardware

Cachés multi-banco

- **Objetivo**: Permitir accesos simultáneos a distintas posiciones de la caché.
- **Solución**: Dividir la memoria en bancos independientes.
- **Efecto**: Aumenta el ancho de banda.
- **Ejemplo**: Sun Niagara
  - L2: 4 bancos

Ancho de banda

- Para que el ancho de banda mejore hace falta que los accesos queden distribuidos entre los bancos.

- **Esquema sencillo**: Entrelazado secuencial.
  - Reparto secuencial de bloques entre los bancos.

Palabra crítica primero y reinicio temprano

- **Observación**: Normalmente el procesador necesita una única palabra para poder proseguir.
- **Solución**: No esperar a recibir todo el bloque de memoria.
- **Alternativas**:
  - **Palabra crítica primero**: El bloque se recibe reordenado con la palabra que necesita el procesador al principio.
  - **Reinicio temprano**: El bloque se recibe sin reordenar.
    - En cuanto se recibe la palabra de interés la CPU prosigue.
- **Efectos**: Depende del tamaño de bloque $\rightarrow$ Mejor más grande.

2 Optimizaciones avanzadas

- Cachés pequeñas y simples
- Predicción de vía
- Acceso segmentado a la caché
- Cachés no bloqueantes
- Cachés multi-banco
- Palabra crítica primero y reinicio temprano
- Mezclas en búfer de escritura
- Optimizaciones del compilador
- Lectura adelantada hardware

Búfer de escritura

- Un **búfer de escritura** permite **reducir** la penalización de fallo.
  - Cuando se ha escrito en el búfer el procesador da la escritura por efectuada.
  - Escrituras simultáneas en memoria son más eficientes que una única escritura.

- **Usos**:
  - **Escritura inmediata** (*write-through*): En todas las escrituras.
  - **Post escritura** (*write-back*): Cuando se remplaza un bloque.

Mezclas en búfer de escritura

- Si el búfer contiene bloques modificados, se comprueban las direcciones para ver si se puede sobrescribir.

- **Efectos**:
  - Reduce el número de **escrituras en memoria**.
  - Reduce **paradas** debidas a que el búfer esté lleno.

Mezclas en buffer de escritura

Write address	V		V		V		V	
100	1	M[100]	0		0		0	
108	1	M[108]	0		0		0	
116	1	M[116]	0		0		0	
124	1	M[124]	0		0		0	

Write address	V		V		V		V	
100	1	M[100]	0	M[108]	0	M[116]	0	M[124]
	1		0		0		0	
	1		0		0		0	
	1		0		0		0	

2 Optimizaciones avanzadas

- Cachés pequeñas y simples
- Predicción de vía
- Acceso segmentado a la caché
- Cachés no bloqueantes
- Cachés multi-banco
- Palabra crítica primero y reinicio temprano
- Mezclas en búfer de escritura
- Optimizaciones del compilador
- Lectura adelantada hardware

Optimizaciones del compilador

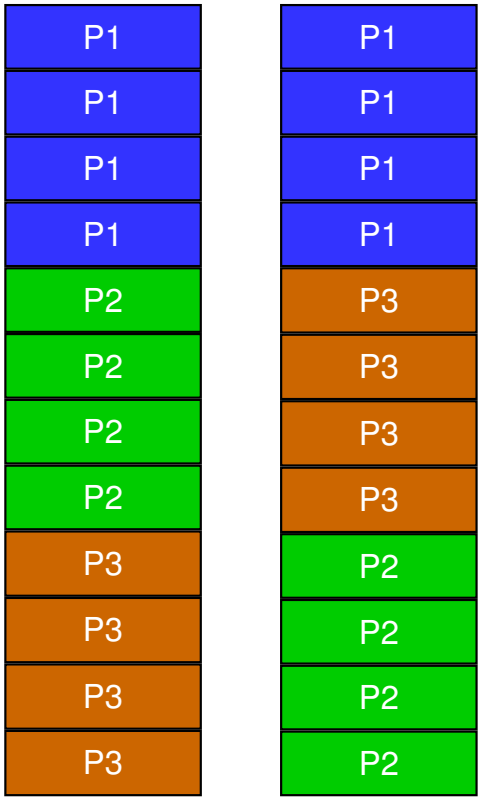
- **Objetivo**: Generar código que provoque menos fallos de instrucciones y datos.

- **Instrucciones**:
  1. Reordenación de procedimientos.
  2. Alinear bloques de código al inicio de bloques de caché.
  3. Linearización de saltos.

- **Datos**:
  1. Fusión de arrays.
  2. Intercambio de bucles.
  3. Fusión de bucles.
  4. Acceso por bloques.

Reordenación de procedimientos

- **Objetivo**: Reducir los **fallos por conflicto** debidos a que dos procedimientos coincidentes en el tiempo se corresponden con la **misma línea de caché**.
- **Técnica**: Reordenar los procedimientos en memoria.



Alineación de bloques básicos

- **Definición**: Un **bloque básico** es un conjunto de instrucciones que se ejecuta secuencialmente (no contiene saltos).
- **Objetivo**: Reducir la posibilidad de **fallos de caché** para código secuencial.
- **Técnica**: Hacer coincidir la **primera instrucción** de un bloque básico con la **primera palabra** de una línea de caché.

Linearización de saltos

- **Objetivo**: Reducir los fallos de caché debidos a saltos condicionales.

- **Técnica**: Si el compilador sabe que es probable que se tome un salto, puede cambiar el sentido de la condición en intercambiar los bloques básicos de las dos alternativas.
  - Algunos compiladores definen extensiones para dar pistas al compilador.
  - **Ejemplo**: **gcc** (**__likely__**).

Fusión de arrays

Arrays paralelos

```
vector<int> key;
vector<int> val;

for (int i=0;i<max;++i) {
  cout << key[i] << ","
       << val[i] << endl;
}
```

Array fusionado

```
struct entry {
  int key;
  int val;
};
vector<entry> v;

for (int i=0;i<max;++i) {
  cout << v[i].key << ","
       << v[i].val << endl;
}
```

- Reducción de conflictos.
- Mejora de localidad espacial.

Intercambio de bucles

Acceso con saltos

```
for (int j=0; j<100; ++j) {
  for (int i=0; i<5000; ++i) {
    v[i][j] = k * v[i][j];
  }
}
```

Accesos secuenciales

```
for (int i=0; i<5000; ++i) {
  for (int j=0; j<100; ++j) {
    v[i][j] = k * v[i][j];
  }
}
```

- **Objetivo**: Mejorar localidad espacial.
- Dependiente del modelo de almacenamiento vinculado al lenguaje de programación.
  - FOTRAN versus C.

Fusión de bucles

Bucles separados

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

Bucles fusionados

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

- **Objetivo**: Mejorar localidad temporal.
- **Cuidado**: Puede reducir localidad espacial.

Acceso por bloques

Producto original

```
for (int i=0; i<size; ++i) {
  for (int j=0; j<size; ++j) {
    r=0;
    for (int k=0; k<size; ++k) {
      r+= b[i][k] * c[k][j];
    }
    a[i][j] = r;
  }
}
```

Producto con acceso por bloques

```
for (bj=0; bj<size; bj+=bsize) {
  for (bk=0; bk<size; bk +=bs) {
    for (i=0; i<size; ++i) {
      for (j=bj; j<min(bj+bsize,size); ++j) {
        r=0;
        for (k=bk;k<min(bk+bsize,size); ++k) {
          r +=b[i][k] * c[k][j];
        }
        a[i][j] += r;
      }
    }
  }
}
```

- **bsize**: Factor de bloque.

2 Optimizaciones avanzadas

- Cachés pequeñas y simples
- Predicción de vía
- Acceso segmentado a la caché
- Cachés no bloqueantes
- Cachés multi-banco
- Palabra crítica primero y reinicio temprano
- Mezclas en búfer de escritura
- Optimizaciones del compilador
- Lectura adelantada hardware

Lectura adelantada de instrucciones

- **Observación**: Las instrucciones presentan alta localidad espacial.
- **Instruction prefetching**: Lectura adelantada de instrucciones.
  - Lectura de dos bloques en caso de fallo.
    - Bloque que provoca el fallo.
    - Bloque siguiente.
- **Ubicación**:
  - Bloque que provoca el fallo → **caché de instrucciones**.
  - Bloque siguiente → **búfer de instrucciones**.

Lectura adelantada de datos

- **Ejemplo**: Pentium 4.
- **Data prefetching**: Permite lectura adelantada de páginas de 4KB a caché L2.
- Se invoca lectura adelantada si:
  - 2 fallos en L2 debidos a una misma página.
  - Distancia entre fallos menor que 256 bytes.

Resumen (I)

- **Cachés pequeñas y simples**:
  - **Mejora**: Tiempo de acierto.
  - **Empeora**: Tasa de fallos.
  - **Complejidad**: Muy baja.
  - **Observación**: Ampliamente usada.
- **Predicción de vía**:
  - **Mejora**: Tiempo de acierto.
  - **Complejidad**: Baja.
  - **Observación**: Usada en Pentium 4.
- **Acceso segmentado a la caché**:
  - **Mejora**: Tiempo de acierto.
  - **Empeora**: Ancho de banda.
  - **Complejidad**: Baja.
  - **Observación**: Ampliamente usada.

Resumen (II)

- **Cachés no bloqueantes**:
  - **Mejora**: Ancho de banda y penalización por fallo.
  - **Complejidad**: Muy alta.
  - **Observación**: Ampliamente usada.
- **Cachés multi-banco**:
  - **Mejora**: Ancho de banda.
  - **Complejidad**: Baja.
  - **Observación**: Usada en L2 de Intel i7 y Cortex A8.
- **Palabra crítica primero y reinicio temprano**:
  - **Mejora**: Penalización por fallo.
  - **Complejidad**: Alta
  - **Observación**: Ampliamente usada.

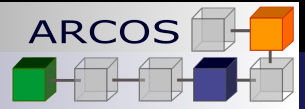
Resumen (III)

- **Mezclas en búfer de escritura**:
  - **Mejora**: Penalización por fallo.
  - **Complejidad**: Baja.
  - **Observación**: Ampliamente usada.
- **Optimizaciones del compilador**:
  - **Mejora**: Tasa de fallos.
  - **Complejidad**: Baja para HW.
  - **Observación**: El desafío está en el software.
- **Lectura adelantada hardware**:
  - **Mejora**: Penalización por fallo y tasa de fallos.
  - **Complejidad**: Muy alta.
  - **Observación**: Más común para instrucciones que datos.

Referencias

- **Computer Architecture. A Quantitative Approach**
  5th Ed.
  Hennessy and Patterson.
  Secciones: 2.1, 2.2.

- **Ejercicios recomendados**:
  - 2.1, 2.2, 2.3, 2.8, 2.9, 2.10, 2.11, 2.12



Optimizaciones avanzadas de memoria caché

Arquitectura de Computadores

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

Grupo ARCOS
Departamento de Informática
Universidad Carlos III de Madrid