Computación de Altas Prestaciones 2022-2023

Jerarquía de caches

Algoritmos a bloques

Cache, conceptos básicos

Definición de cache:small high-speed buffer memory between the processor and main memory.

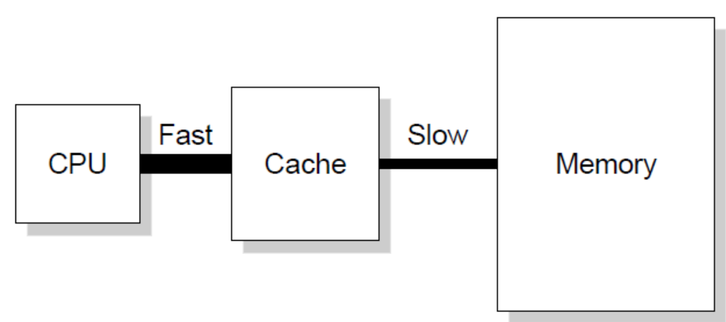

FIGURE 1 Example of a Cache-Based Memory System.

El concepto de cache es clave para el rendimiento de los ordenadores modernos.

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Cache, conceptos básicos

-línea de cache

-Cache “write-through” o “write-back”

-Niveles de cache (suele haber al menos dos)

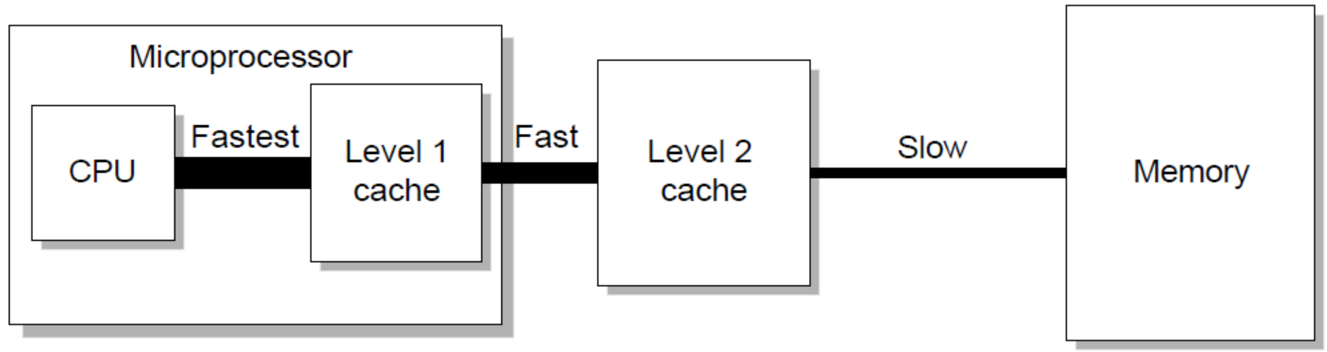

FIGURE 2 Multiple Levels of Cache Memory

-Latencia, Ancho de Banda

Cache, conceptos básicos

-Memoria Virtual: Cache para las direcciones físicas de las páginas de memoria: Translation Lookaside Buffer (TLB).

Organización básica:

Nivel 1: cache de datos, cache de Instrucciones y cache TLB independientes

Nivel 2: cache de datos, de instrucciones y TLB unificadas

Cache, conceptos básicos

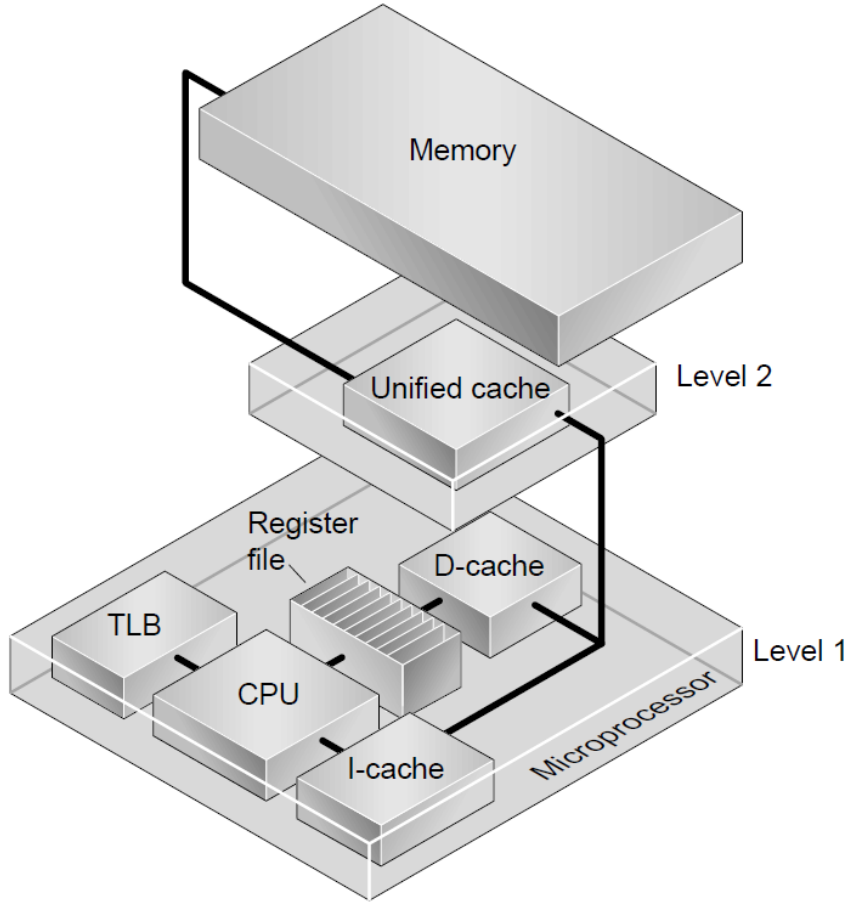

FIGURE 3 Generic System Architecture

DSIC
DEPARTAMENTO DE SISTEMAS
INFORMÁTICOS Y COMPUTACIÓN

Prefetch

Podemos mejorar el rendimiento si podemos anticipar que datos/instrucciones/direcciones va a necesitar nuestro programa

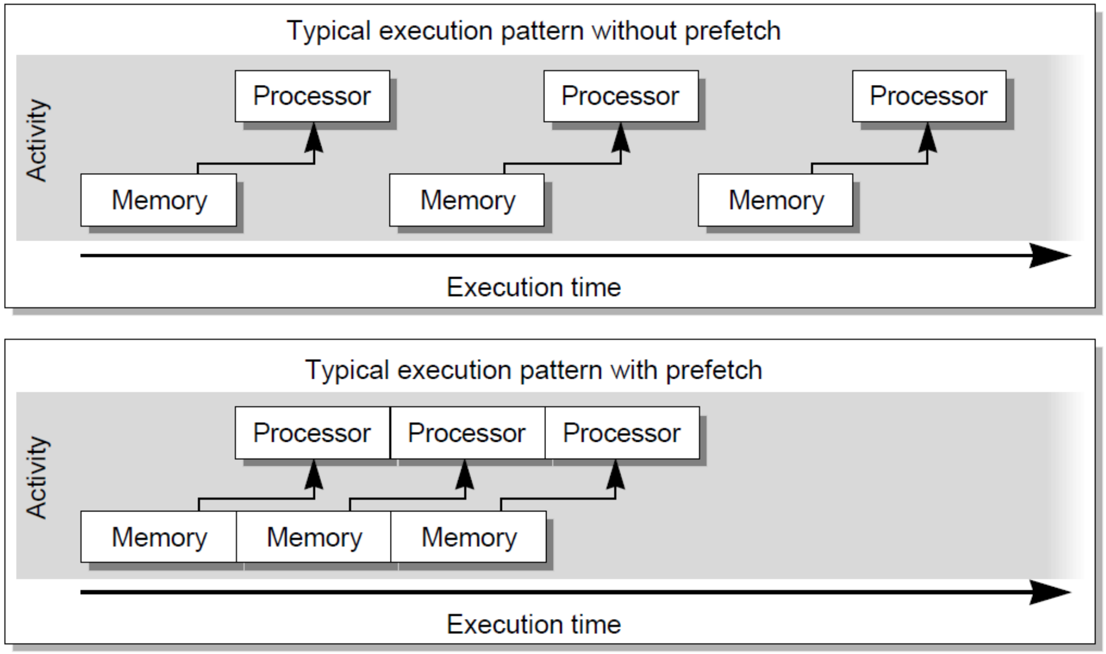

FIGURE 4 Prefetch

Organización de cache y políticas de reemplazo

Para cada nuevo acceso a memoria, se debe decidir en que línea de la cache se debe guardar el nuevo bloque de memoria.

1) Direct Map: La dirección de memoria de la línea de cache que se tiene que traer, dicta en que línea de la cache se va a guardar.

Problema de Direct Map: Cache Thrashing

Es posible que se produzca un patrón de acceso a memoria poco favorable, en el que se produzcan repetidos fallos de cache

Organización de cache y políticas de reemplazo

Ejemplo: cache de 4K con líneas de 32 bytes; Una variable float usaría 4 bytes y caben 8 en una línea.

Si ejecutamos este código:

```
float a[1024], b[1024];
for (i=0; i<1024; i++)
    sum += a[i]*b[i];
```

Si a y b están almacenados contiguamente en memoria, este código provocará fallos de cache en cada acceso.

Paso Crítico (Critical Stride): 4K

Organización de cache y políticas de reemplazo

Patrón de acceso si a y b están “separados” en memoria, acceso óptimo:

1) Se trae a[0] a cache => fallo => se traen a[0],...,a[7]
2) Se trae b[0] a cache=> fallo => se traen b[0],...,b[7]
3) Se trae a[1] a cache=> Ya está, no se hace nada
4) Se trae b[1] a cache=> Ya está, no se hace nada

...

17) Se trae a[8] a cache => fallo => se traen a[8],...,a[15]
18) Se trae b[8] a cache=> fallo => se traen b[8],...,b[15]
19) Se trae a[9] a cache=> Ya está, no se hace nada

Un fallo cada 8 accesos

Organización de cache y políticas de reemplazo

Patrón de acceso si a y b están “consecutivos” en memoria, acceso pésimo: Ambos van a la misma línea de cache

1) Se trae a[0] a cache => fallo => se traen a[0],...,a[7]
2) Se trae b[0] a cache=> fallo => se traen b[0],...,b[7]; se sobreescriben sobre a[0],...,a[7]
3) Se trae a[1] a cache=> fallo => se traen a[0],...,a[7]]; se sobreescriben sobre b[0],...,b[7]
4) Se trae b[1] a cache=> fallo => se traen b[0],...,b[7]; se sobreescriben sobre a[0],...,a[7]

etc.

Cada acceso es un fallo

Organización de cache y políticas de reemplazo

Posible solución : loop unrolling:

```
for (i=0; i<1024; i+=2){
        ta0 = a[i];
        ta1 = a[i+1];
        tb0 = b[i];
        tb1 = b[i+1];
        sum += ta0*tb0+ta1*tb1;
}
```

Problema: se necesitan mas registros.

Organización de cache y políticas de reemplazo

Mejor todavía separar los vectores en memoria:

```
float a[1024], hueco[8], b[1024];
for (i=0; i<1024; i++)
        sum += a[i]*b[i];
```

Organización de cache y políticas de reemplazo

Política 2: Fully Associative (LRU); Muy costosa

Política 3: Set Associative

La cache se divide en conjuntos; Cada conjunto usa internamente política DirectMap; La decisión de a que conjunto va cada linea se hace mediante LRU, aleatorio o cualquier otro esquema.

"paso crítico", o "Critical Stride": nº ways* tamaño conjuntos

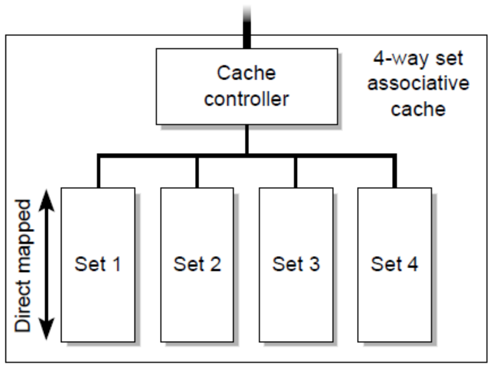

FIGURE 5 Four-Way Set Associative Design.

Computación de Altas Prestaciones 2022-2023

Algoritmos a bloques

Analisis del producto de matrices , orden j, k, i

```
For j=1:n
        For k=1:n
                For i=1:n
                        C(i,j)=C(i,j)+A(i,k)*B(k,j)
                End
        End
End
```

-Este algoritmo es uno de los mejores , si no el mejor, si la matriz está organizada por columnas.

-Además el bucle mas externo, el j ,es paralelizable . Cada iteración del bucle j es un producto matriz vector, usando la columna j de B y guardando en la columna j de C

C(:,1)=C(:,1)+A*B(:,1)

C(:,2)=C(:,2)+A*B(:,2)

...

Analisis del producto de matrices , orden j, k, i

```
For j=1:n
        For k=1:n
                For i=1:n
                        C(i,j)=C(i,j)+A(i,k)*B(k,j)
                End
        End
End
```

Desde el punto de vista del número de accesos a memoria, la matriz A se carga entera en cada iteración del bucle j.

Convendría poder reusar mas los datos, de forma que no se tuviera que cargar cada dato n veces.

Para lograr esto, se utilizan los algoritmos por bloques.

Objetivo de los algoritmos a bloques

Los algoritmos a bloques son variaciones de algoritmos tradicionales, en los que un problema grande se descompone en trozos más pequeños que “caben” en la memoria cache. Se usan para problemas “Grandes”.

Seleccionando cuidadosamente el tamaño de los bloques y el algoritmo, se puede minimizar el tráfico de memoria necesario para resolver el problema.

Todas las implementaciones en librerías de algebra lineal numérica (solución de Sistemas de Ecuaciones Lineales, cálculo de valores y vectores propios, solución de problemas de mínimos cuadrados, etc.) están hechas con Algoritmos a bloques, basados en el producto de matrices.

Estructura general de los algoritmos a bloques

En los algoritmos típicos del algebra lineal numérica, las versiones a bloques tienen una estructura común. Supondremos que tenemos un solo nivel de cache. Las implementaciones “rápidas” contemplan tantos niveles de “bloques” como niveles de cache haya disponibles.

Sea P el problema que queremos resolver; (Descomposición LU, Descomposición Cholesky, resolución de sistemas triangulares, Descomposición QR, etc.)

Estructura general de los algoritmos a bloques

Para obtener la versión orientada a bloques del algoritmo P, necesitaremos:

-Una versión del algoritmo P para matrices “que quepan en la cache”; Dicho de otro modo, una versión Pv para matrices “pequeñas”.

-Una subrutina auxiliar de producto de matrices a bloques

-En algún caso, también subrutinas para resolver sistemas triangulares a bloques

Habitualmente se puede reestructurar el algoritmo P de forma que solo contenga llamadas a la subrutina Pv y a las subrutinas auxiliares

Notación-1

Sea A $\in \Re^{m*n}$; es posible partir A de la siguiente forma:

$$A = \begin{array}{cl} \begin{pmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,r} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,r} \\ \vdots & \vdots & \ddots & \vdots \\ A_{q,1} & A_{q,2} & \cdots & A_{q,r} \end{pmatrix} & \begin{array}{c} m_1 \\ m_2 \\ \\ m_q \end{array} \\ \begin{array}{cccc} n_1 & n_2 & \quad & n_r \end{array} & \end{array}$$

Donde cada bloque $A_{i,j}$ tiene dimensión $m_i \times n_{j;}$

$m_1 + m_2 + \ldots + m_q = m$ $n_1 + n_2 + \ldots + n_q = n$

Notación-2

Si B $\in \Re^{m*n}$:

$$B = \begin{array}{cc} \begin{pmatrix} B_{1,1} & B_{1,2} & \cdots & B_{1,r} \\ B_{2,1} & B_{2,2} & \cdots & B_{2,r} \\ \vdots & \vdots & \ddots & \vdots \\ B_{q,1} & B_{q,2} & \cdots & B_{q,r} \end{pmatrix} & \begin{array}{c} m_1 \\ m_2 \\ \\ m_q \end{array} \\ \begin{array}{cccc} n_1 & n_2 & \quad & n_r \end{array} & \end{array}$$

La partición de B está **conforme** con la de la matriz A (diapositiva anterior)

Notación-3

La suma de C=A+B también será una matriz a bloques con idéntica partición:

$$C = A + B = \begin{pmatrix} C_{1,1} & C_{1,2} & \cdots & C_{1,r} \\ C_{2,1} & C_{2,2} & \cdots & C_{2,r} \\ \vdots & \vdots & \ddots & \vdots \\ C_{q,1} & C_{q,2} & \cdots & C_{q,r} \end{pmatrix} = \begin{pmatrix} A_{1,1}+B_{1,1} & A_{1,2}+B_{1,2} & \cdots & A_{1,r}+B_{1,r} \\ A_{2,1}+B_{2,1} & A_{2,2}+B_{2,2} & \cdots & A_{2,r}+B_{2,r} \\ \vdots & \vdots & \ddots & \vdots \\ A_{q,1}+B_{q,1} & A_{q,2}+B_{q,2} & \cdots & A_{q,r}+B_{q,r} \end{pmatrix}$$

Producto de Matrices-1

Caso general: A $\in \Re^{m*p}$, B $\in \Re^{p*n}$

$$A = \begin{array}{c} \begin{pmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,s} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,s} \\ \vdots & \vdots & \ddots & \vdots \\ A_{q,1} & A_{q,2} & \cdots & A_{q,s} \end{pmatrix} \\ \begin{matrix} p_1 & p_2 & \cdots & p_s \end{matrix} \end{array} \qquad B = \begin{pmatrix} B_{1,1} & B_{1,2} & \cdots & B_{1,r} \\ B_{2,1} & B_{2,2} & \cdots & B_{2,r} \\ \vdots & \vdots & \ddots & \vdots \\ B_{s,1} & B_{s,2} & \cdots & B_{s,r} \end{pmatrix} \begin{matrix} p_1 \\ p_2 \\ \vdots \\ p_s \end{matrix}$$

$$A \cdot B = C = \begin{bmatrix} C_{1,1} & C_{1,2} & \cdots & C_{1,r} \\ C_{2,1} & C_{2,2} & \cdots & C_{2,r} \\ \vdots & \vdots & \ddots & \vdots \\ C_{q,1} & C_{q,2} & \cdots & C_{q,r} \end{bmatrix};$$

Dimensiones

Donde $\boxed{C_{\alpha,\beta} = \sum_{j=1}^{s} A_{\alpha,j} \cdot B_{j,\beta}}$

Producto (Matriz a Bloques)·Vector

Gaxpy, $y=Ax+y$: $A \in \Re^{m*n}$, $x \in \Re^{n}$, $y \in \Re^{m}$

Versión con A partida por filas:

$$A = \begin{pmatrix} A_1 \\ \vdots \\ A_q \end{pmatrix}; \quad \begin{array}{l} m_1 \\ \\ m_q \end{array} \qquad y = \begin{pmatrix} y_1 \\ \vdots \\ y_q \end{pmatrix} \quad \begin{array}{l} m_1 \\ \\ m_q \end{array}$$

Sea **vec** el vector de "longitudes" de los bloques:
Vec=$(m_1, m_2, \ldots, m_q)$

Producto (Matriz a Bloques) ·Vector

$$\begin{pmatrix} y_1 \\ \vdots \\ y_q \end{pmatrix} = \begin{pmatrix} A_1 \\ \vdots \\ A_q \end{pmatrix} x + \begin{pmatrix} y_1 \\ \vdots \\ y_q \end{pmatrix};$$

Algoritmo

```
Last=0
For i=1:q
        first=last+1
        last=first+vec(i)-1
        y(first:last)=A(first:last,:)*x+y(first:last)
End For
```

Gaxpy de
un bloque

Producto matrices a bloques

$$C_{i,j} = \sum_{k=1}^{s} A_{i,k} \cdot B_{k,j} + C_{i,j}$$

Necesitamos una función Pv (pequeña), para hacer el producto de matrices que quepan en la cache:

Vamos a llamarla matmul_cache:

Producto matrices a bloques

A Matmul_cache se le pasan tres bloques C, A y B, y calcula C=C+A*B

```
For i=1:N        { N es el número de bloques}
   ind_i=(i-1)*tb+1:i*tb
   For j=1:N
       ind_j=(j-1)*tb+1:j*tb
            For k=1:N
                ind_k=(k-1)*tb+1:k*tb
                matmul_cache(C(ind_i,ind_j), A(ind_i,ind_k),B(ind_k,ind_j))
            End
   End
End
```

Es preciso tener en cuenta que son "bloques" contenidos en las matrices "grandes" A, B y C.