Tabla 4.7. Características de las memorias de dos niveles.

	Caché de memoria principal	Memoria virtual (paginación)	Caché de disco
Relaciones de tiempos de acceso típicas	5/1	$10^6 : 1$	$10^6 : 1$
Sistema de gestión de memoria	Implementado en hardware especial	Combinación de hardware y software del sistema	Software de sistema
Tamaño del bloque típico	4 a 128 bytes	64 a 4.096 bytes	64 a 4.096 bytes
Acceso del procesador al segundo nivel	Acceso directo	Acceso indirecto	Acceso indirecto

memoria virtual se verá en el Capítulo 8; la caché de disco queda fuera del alcance de este libro pero es examinada en [STAL05]. En este apéndice veremos algunas características de prestaciones de las memorias de dos niveles que son comunes a las tres aproximaciones mencionadas.

LOCALIDAD

La base para la mejora de prestaciones de una memoria de dos niveles es un principio conocido como **localidad de las referencias** [DENN68]. Este principio establece que las referencias a memoria tienden a formar agrupaciones (*clusters*). A lo largo de un período de tiempo largo las agrupaciones en uso cambian, pero durante periodos cortos el procesador trabaja fundamentalmente con agrupaciones fijas de referencias a memoria.

Intuitivamente, el principio de localidad tiene sentido. Considérense la siguiente secuencia de razonamientos:

1. Excepto por las instrucciones de bifurcación y de llamada, que constituyen solo una pequeña fracción de todas las instrucciones, la ejecución de un programa es secuencial. Por tanto, en la mayoría de los casos, la siguiente instrucción a captar sigue inmediatamente a la última captada.
2. Es raro tener una secuencia larga ininterrumpida de llamadas a procedimientos seguidas por la correspondiente secuencia de retornos. En su lugar, un programa queda confinado a una ventana bastante estrecha de profundidad o nivel de anidamiento de procedimientos. Así pues, a lo largo de un periodo de tiempo corto las referencias a instrucciones tienden a localizarse en unos cuantos procedimientos.
3. La mayoría de las construcciones iterativas constan de un número relativamente pequeño de instrucciones repetidas muchas veces. Durante una iteración, el procesamiento está por tanto confinado a una pequeña porción contigua del programa.
4. En muchos programas, gran parte del cálculo incumbe al procesamiento de estructuras de datos, tales como matrices o secuencias de registros. En muchos casos, las referencias sucesivas a estas estructuras de datos serán a unidades de datos ubicados próximos entre sí.

Esta secuencia de razonamientos ha sido confirmada en muchos estudios. En relación al punto 1, se han hecho diversos estudios para analizar el comportamiento de programas en lenguajes de alto nivel. La Tabla 4.8 recoge, de los estudios que se indican, resultados clave que miden la aparición de

Tabla 4.8. Frecuencia dinámica relativa de operaciones en lenguajes de alto nivel.

Estudio	[HUCK83]	[KNUT71]	[PATT82a]		[TANE78]
Lenguaje	Pascal	FORTRAN	Pascal	C	SAL
Tipo de trabajo	Científico	Estudiante	Sistema	Sistema	Sistema
Assign	74	67	45	38	42
Loop	4	3	5	3	4
Call	1	3	15	12	12
IF	20	11	29	43	36
GOTO	2	9	—	3	—
Otras	—	7	6	1	6

distintos tipos de sentencias durante la ejecución. El primero de los estudios sobre el comportamiento de lenguajes de programación, realizado por Knuth [KNU71], evaluó un conjunto de programas FORTRAN utilizados como ejercicios de estudiantes. Tanenbaum [TANE78] publicó medidas recopiladas de más de trescientos procedimientos utilizados en programas de sistemas operativos y escritos en un lenguaje que soporta programación estructurada (SAL). Patterson y Sequin [PATT82a] analizaron un conjunto de medidas tomadas de compiladores y programas para composición de textos, CAD, ordenación, y comparación de ficheros. Se estudiaron los lenguajes de programación C y Pascal. Huck [HUCK83] analizó cuatro programas ideados para una mezcla de cálculos científicos de uso general, incluyendo la transformada rápida de Fourier y la resolución de sistemas de ecuaciones diferenciales. Hay bastante coincidencia, en los resultados de esta mezcla de lenguajes y de aplicaciones, en que las instrucciones de bifurcación y de llamada representan solo una fracción de las sentencias ejecutadas durante el tiempo de vida de un programa. Estos estudios confirman pues la afirmación 1 anterior.

Con respecto a la afirmación 2, estudios aportados en [PATT85a] la confirman. Esto se ilustra en la Figura 4.16, que muestra el comportamiento de la pareja llamada-retorno. Cada llamada es representada mediante la línea hacia abajo y hacia la derecha, y cada retorno mediante la línea hacia arriba y a la derecha. En la figura se ha definido una *ventana* con profundidad igual a 5. Solo una secuencia de llamadas y retornos con variación de 6 en cualquier dirección hace que se traslade la

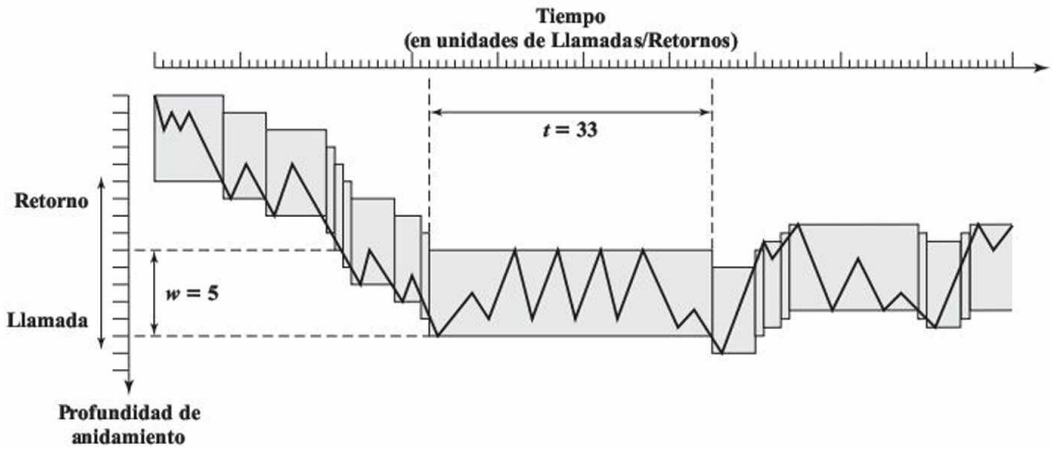

Figura 4.16 . Ejemplo de comportamiento de las llamadas/retornos en un programa.

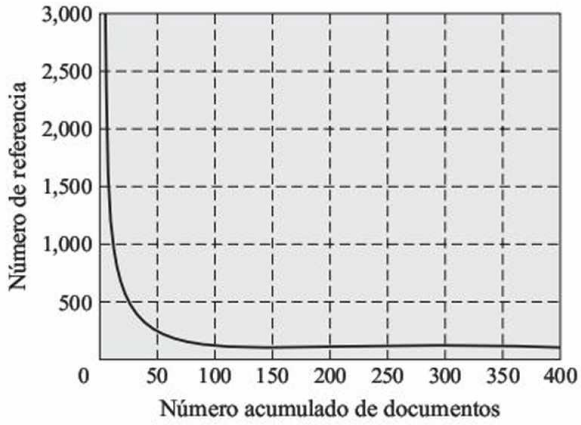

Figura 4.17. Localidad de referencias para páginas Web [BAEN97].

ventana. Como puede verse, el programa en ejecución puede mantenerse dentro de una ventana estacionaria por periodos de tiempo bastante largos. Un estudio por los mismos analistas de programas en C y en Pascal mostró que una ventana de profundidad 8 solo necesitaría desplazarse en menos del 1 por ciento de las llamadas o retornos [TAMI83].

El principio de localidad de las referencias continúa siendo validado en estudios más recientes. Por ejemplo, la Figura 4.17 muestra los resultados de un estudio sobre patrones de acceso a páginas web en un mismo sitio web.

En la literatura se distingue entre localidad espacial y temporal. La **localidad espacial** se refiere a la tendencia durante la ejecución a involucrar múltiples posiciones de memoria que estén agrupadas. La localidad espacial refleja la tendencia del procesador a acceder a las instrucciones secuencialmente y también la tendencia de los programas a acceder a posiciones de datos consecutivas, como por ejemplo cuando se procesa una tabla de datos. La **localidad temporal** hace referencia a la tendencia del procesador a acceder a posiciones de memoria que han sido utilizadas recientemente. Por ejemplo, cuando se ejecutan iteraciones de un bucle, el procesador ejecuta repetidamente el mismo conjunto de instrucciones.

Tradicionalmente, la localidad temporal se ha explotado manteniendo en memoria caché las instrucciones y los datos usados recientemente, y aprovechando la jerarquía de caché. Generalmente, la localidad espacial se explota usando bloques de caché más grandes e incorporando mecanismos de precaptación (captando objetos de uso anticipado) en la lógica de control de caché. Recientemente se ha investigado bastante para refinar estas técnicas al objeto de conseguir mayores prestaciones, pero las estrategias básicas siguen siendo las mismas.

FUNCIONAMIENTO DE LA MEMORIA DE DOS NIVELES

La propiedad de localidad puede ser aprovechada formando una memoria de dos niveles. La memoria del nivel superior (M1) es más pequeña más rápida, y más costosa (por bit) que la del nivel inferior (M2). M1 se utiliza como almacén temporal para una parte del contenido de la otra más grande, M2. Cuando se hace una referencia a memoria, se intenta acceder al elemento en M1. Si

tiene éxito, entonces tiene lugar un acceso rápido. Si no, se copia un bloque de posiciones de memoria de M2 a M1, y el acceso se hace vía M1. Debido a la localidad, una vez que el bloque es llevado a M1 habrá un número de accesos a posiciones de ese bloque, resultando un servicio rápido en su conjunto.

Para expresar el tiempo medio de acceso a un elemento, debemos considerar no solo las velocidades de los dos niveles de memoria, sino también la probabilidad de que una referencia dada se encuentre en M1. Tenemos:

$$\begin{aligned} T_s &= H \times T_1 + (1 - H) \times (T_1 + T_2) \\ &= T_1 + (1 - H) \times T_2 \end{aligned} \tag{4.1}$$

donde

T_s = tiempo de acceso medio (del sistema)

T_1 = tiempo de acceso de M1 (por ejemplo: caché, caché de disco)

T_2 = tiempo de acceso de M2 (por ejemplo: memoria principal, disco)

H = tasa de aciertos (fracción de veces que la referencia es encontrada en M1)

La Figura 4.2, al principio del capítulo, muestra el tiempo de acceso medio en función de la tasa de aciertos. Como puede verse, para un porcentaje de aciertos alto el tiempo de acceso total medio es mucho más próximo al de M1 que al de M2.

PRESTACIONES

Veamos algunos parámetros relevantes a la hora de evaluar un esquema de memoria de dos niveles. Consideremos primero el coste. Tenemos:

$$C_s = \frac{C_1 S_1 + C_2 S_2}{S_1 + S_2} \tag{4.2}$$

donde:

C_s = coste medio por bit de la combinación de dos niveles de memoria

C_1 = coste medio por bit de la memoria M1 del nivel superior

C_2 = coste medio por bit de la memoria M2 del nivel inferior

S_1 = tamaño de M1

S_2 = tamaño de M2

Nos gustaría que $C_s \approx C_2$. Dado que $C_1 >> C_2$, ello requiere que $S_1 << S_2$. La Figura 4.18 muestra dicha relación.

Consideremos ahora el tiempo de acceso. Para que una memoria de dos niveles suponga una mejora de prestaciones significativa, necesitamos tener T_s aproximadamente igual a T_1 ($T_s \approx T_1$). Dado que T_1 es mucho menor que T_2 ($T_1 << T_2$), se necesita una tasa de aciertos próxima a 1.

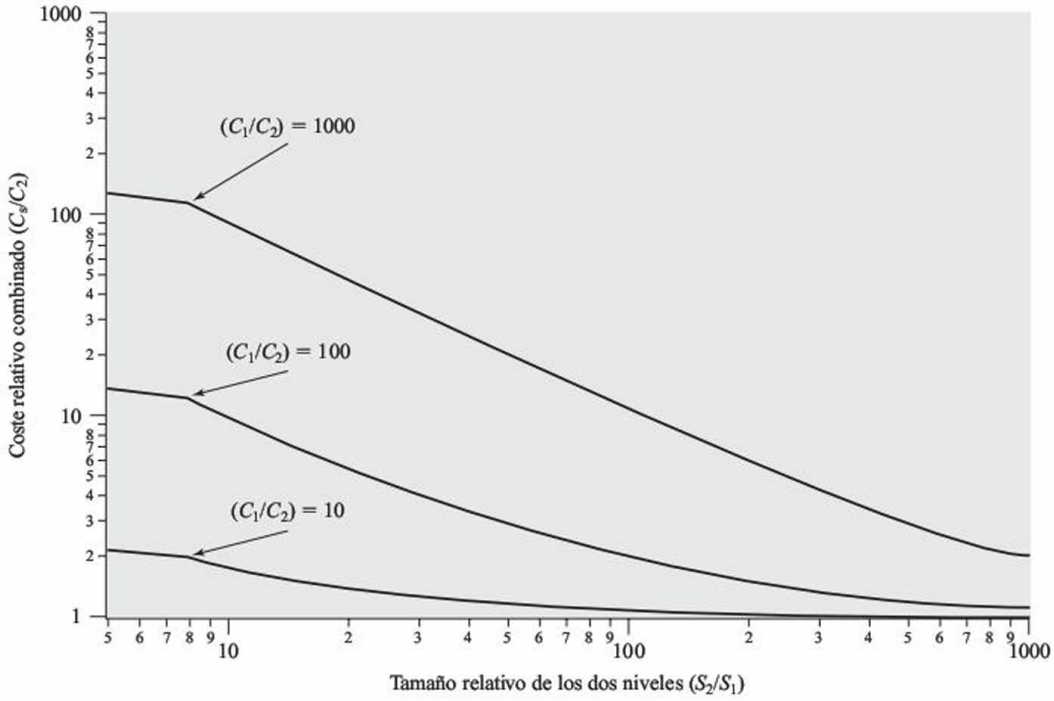

Figura 4.18. Relación entre el coste de memoria medio y el tamaño relativo de las memorias, para una memoria de dos niveles.

Así pues, nos gustaría que M1 fuera pequeña para mantener bajo el coste, y grande para mejorar la tasa de aciertos y en consecuencia las prestaciones. ¿Hay un tamaño de M1 que satisfaga razonablemente ambos requisitos? Esta pregunta la podemos responder con una serie de subpreguntas:

- ¿Qué valor de tasa de aciertos se necesita para que $T_s \approx T_1$?
- ¿Qué tamaño de M1 asegurará la tasa aciertos necesaria?
- ¿Satisface dicho tamaño el requisito del coste?

Para responder, consideremos la cantidad T_1/T_s, conocida como *eficiencia de acceso*. Es una medida de cuán próximo es el tiempo de acceso medio (T_s) al tiempo de acceso de M1 (T_1). De la ecuación (4.1) resulta:

$$\frac{T_1}{T_s} = \frac{1}{1 + (1 - H)\dfrac{T_2}{T_1}} \tag{4.3}$$

En la Figura 4.19 se representa T_1/T_s en función de la tasa de aciertos H, con la cantidad T_1/T_2 como parámetro. Normalmente el tiempo de acceso a la caché *on-chip* es entre 25 y 50 veces menor que el tiempo de acceso a memoria principal (es decir T_2/T_1 está entre 25 y 50), el tiempo de acceso

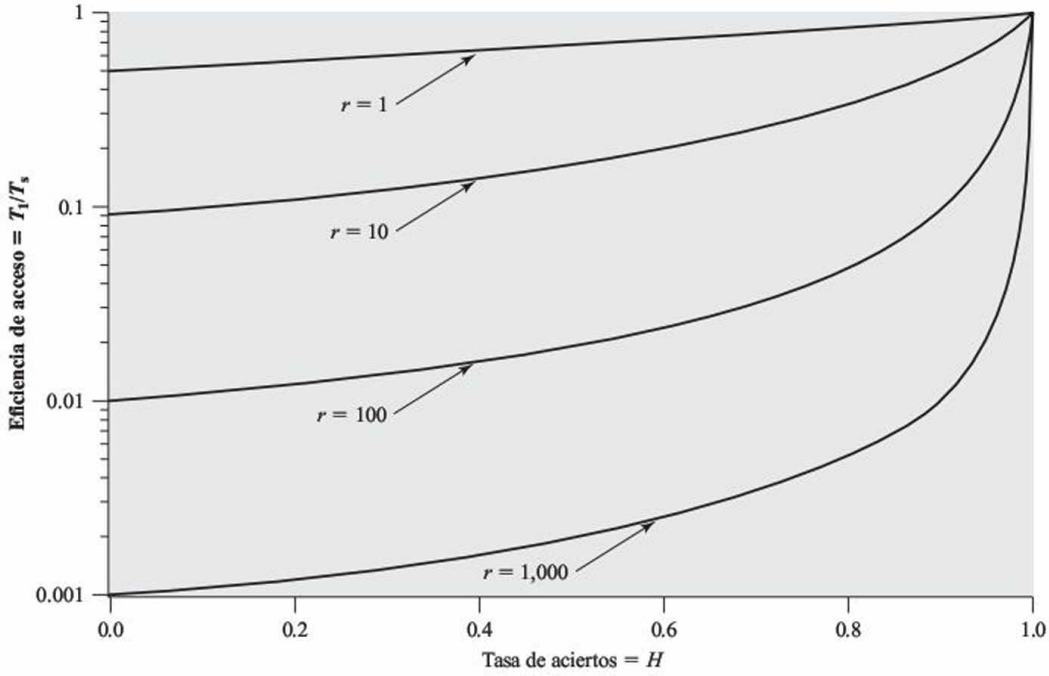

Figura 4.19. Eficiencia de acceso en función de la tasa de aciertos (r = T2/T1).

a caché externa (*off-chip*) es entre cinco y quince veces menor que el tiempo de acceso a memoria principal (es decir T_2/T_1 está entre 5 y 15)[7] y el acceso a memoria principal es del orden de 1 000 veces más rápido que el acceso a disco (T_2/T_1 = 1 000). Por tanto, parece necesaria una tasa de aciertos entre 0,8 y 0,9 para satisfacer el requisito de prestaciones.

Ahora podemos plantear con mayor exactitud la cuestión referida al tamaño relativo de las memorias. ¿Es razonable una tasa de aciertos de 0,8 o superior con $S_1 << S_2$? Eso dependerá de diversos de factores, incluida la naturaleza del software que se ejecute y los detalles del diseño de la memoria de dos niveles. Lo más decisivo es, por supuesto, el grado de localidad. La Figura 4.20 sugiere el efecto que tiene la localidad sobre la tasa de aciertos. Claramente, si M_1 tiene el mismo tamaño que M_2, la tasa de aciertos será 1,0: todos los contenidos de M_2 están también siempre almacenados en M_1. Supongamos ahora que no hay localidad, esto es, que las referencias son completamente aleatorias. En este caso la tasa de aciertos sería una función estrictamente lineal del tamaño relativo de las memorias. Por ejemplo, si M_1 tiene la mitad de capacidad que M_2, en todo momento la mitad de los elementos de M_2 están también en M_1, y la tasa de aciertos será 0,5. En la práctica sin embargo existe cierto grado de localidad en las referencias a memoria. Los efectos de una localidad moderada y fuerte se indican en la figura.

Así pues, si la localidad es fuerte es posible conseguir una tasa de aciertos alta incluso con un tamaño relativamente pequeño de la memoria de nivel superior. Por ejemplo, numerosos estudios han

[7] Por ejemplo, para un Pentium 4 el tiempo de acceso a la caché *on-chip* es de 1 ns para la caché de datos, 2 ns para la caché de instrucciones y 3,5 ns para la caché L2; el tiempo de acceso a la memoria principal es de 30 ns. Para el Itanium 2, el tiempo de acceso a la caché *on-chip* es de 0,67 ns para la caché L1, 4 ns para la caché L2 y 8 ns para la caché L3.

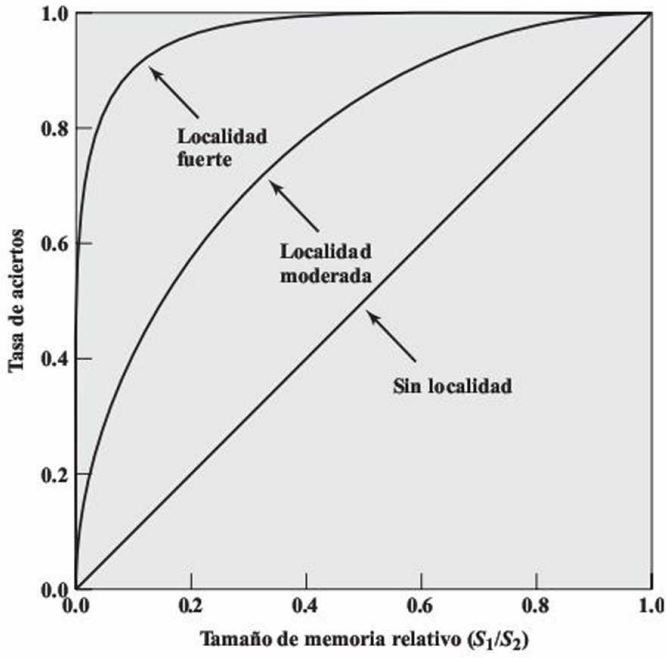

Figura 4.20. Tasa de aciertos en función del tamaño relativo de las memorias.

mostrado que tamaños de caché bastante pequeños producen tasas de aciertos por encima de 0,75, *con independencia del tamaño de la memoria principal* (*consúltese* por ejemplo [AGAR89], [PRZY88], [STRE83], y [SMIT82]). Generalmente es adecuada una caché de 1K a 128K palabras, mientras que una memoria principal actual se mueve en el rango de los gigabytes. Cuando consideremos la memoria virtual y la caché de disco citaremos otros estudios que confirman el mismo fenómeno, es decir que una M_1 relativamente pequeña produce un valor elevado de la tasa de aciertos gracias a la localidad.

Esto nos conduce a la última pregunta antes planteada: ¿satisface el tamaño relativo de las dos memorias el requisito del coste? La respuesta es claramente sí. Si para conseguir buenas prestaciones necesitamos poca capacidad para la memoria de nivel superior, el coste medio por bit de la memoria de dos niveles se aproximará a la del nivel inferior, que es más económica.

Observe que cuando hay implicada caché L2, o incluso cachés L2 y L3, el análisis es mucho más complejo. Consulte [PEIR99] y [HAND98] para una discusión sobre este tema.