Universidad de la Frontera
Facultad de Ingeniería y Administración
Departamento de Ingeniería Eléctrica

ESTRUCTURA DE COMPUTADORES

ALGORITMOS DE REEMPLAZO EN MEMORIAS CACHE

JESÚS RUIZ MATUS

INDICE

1. MEMORIAS CACHE

Las memorias cache fueron introducidas en la década de los 60 en los grandes computadores de la época[1] como una manera de aumentar la velocidad de procesamiento de éstos, superando la diferencia de velocidad existente entre los procesadores y la memoria principal compuesta mayoritariamente en ese instante por núcleos magnéticos, posteriormente, en los '70 se aplicó a los minicomputadores[2] y en los '80 a los microcomputadores.

La memoria cache es una memoria pequeña y rápida, que se inserta entre el procesador y la memoria principal con el objeto que la velocidad de respuesta de la memoria sea lo más parecida a la velocidad de operación del procesador, y sin producir un aumento significativo en el costo del computador. La característica más importante de un sistema que incluya memoria cache es la razón de acierto, esto es, el porcentaje de referencias de memoria que es satisfecho por la memoria cache (Hit ratio). Otra característica importante es que la memoria cache es transparente para el usuario, es decir, el usuario computacional no se entera de la existencia de este nivel de memoria.

Actualmente, las memorias cache se usan en todos los computadores de mediana y alta velocidad para almacenar temporalmente aquellas porciones de memoria que están en uso. Dado que las instrucciones y datos en la memoria cache pueden ser accesados en un 10 - 20 % del tiempo requerido para accesar la memoria principal, el uso de memoria cache permite que el tiempo de ejecución de la máquina se vea sustancialmente disminuido. Por otro lado, el uso de memorias cache, y en general cualquier esquema de jerarquización de memoria se basa en la propiedad de localidad de los programas; esto es, si en un momento dado se hace referencia a una localidad cualquiera de memoria, es muy probable que en el futuro se haga referencia a la misma localidad o a una localidad cercana.

---

[1] Liptay, J. S. Structural aspects of the System/360 Model 85, II The cache 'IBM System Journal' vol 7,1 (1968) 15-21.

[2] Strecker, W.D. Cache Memories for PDP-11 Family Computers. Proceedings of Computer Architecture Symposium, (1976) 155-158.

Estos son a grosso modo los aspectos generales de una memoria cache, que con el objeto de funcionar efectivamente debe ser cuidadosamente diseñada e implementada.

Aún cuando el objetivo del presente trabajo es el estudio de los algoritmos de reemplazo en memorias cache, se verán someramente algunos otros aspectos de importancia en el diseño de memoria cache como son:

- Tamaño de memoria cache
- Mapeo de memoria cache
- Tamaño de bloque y conjunto
- Algoritmo de búsqueda

2. ASPECTOS IMPORTANTES EN EL DISEÑO DE MEMORIAS CACHE

El primer aspecto a considerar en el diseño de una memoria cache es el tamaño que tendrá ésta. Para ello se hacen simulaciones de la carga de trabajo que se supone tendrá el computador diseñado y analizándose la razón de acierto para cada tamaño de memoria cache considerado. Un gráfico típico de esta simulación se muestra en la *Figura 1.*

A partir de este gráfico se determina el tamaño de memoria cache que dé un óptimo para la relación razón de acierto / costo, quedando así resuelto el primer aspecto del diseño de la memoria cache.

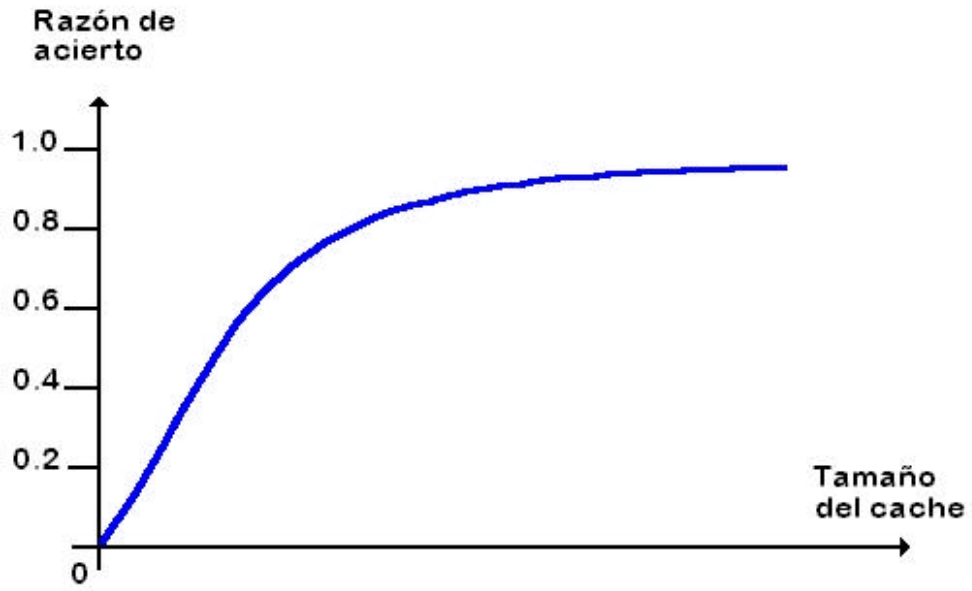

Figura 1.

El siguiente aspecto a considerar en el diseño es el esquema de mapeo que se utilizará. Los esquemas típicos son: mapeo directo, mapeo totalmente

asociativo y mapeo asociativo por conjuntos; donde el mapeo totalmente asociativo es aquel en que el número de conjuntos es uno y el mapeo directo como el caso en que el tamaño del conjunto es uno, por lo cual en el diseño actual de memorias cache se considera solo el mapeo asociativo por conjuntos.

La determinación del tamaño óptimo del bloque y del conjunto requiere nuevamente simulaciones que representen la carga de trabajo del computador. El procedimiento usado consiste en fijar uno de estos parámetros y a través de las simulaciones obtener el valor óptimo para el otro parámetro; con este óptimo fijo debe determinarse el valor óptimo para el primer parámetro, repitiendo si fuere necesario.

Considerando el importante efecto que tienen en el diseño de una memoria cache las simulaciones a que se hace referencia en los párrafos anteriores, es conveniente explicar brevemente cómo se consiguen. Por lo general, se simula la operación de la maquina y se almacenan en un archivo adhoc a las referencias de memoria generadas con el objeto de analizarlas y estimar su comportamiento.

Uno de los aspectos más importantes consiste en la selección del conjunto de programas que se ejecutan en la maquina simulada. Es importante que tal conjunto sea representativo de la carga de trabajo que se supone tendrá el computador; como esto no se conoce a priori, los programas a considerar deben ser de distinto tipo, en diferentes lenguajes, y suficientemente extensos de modo que en total las referencias a memoria sean del orden de un millón o más según recomiendan diferentes autores[3] [4] [5]. En las referencias citadas puede encontrarse abundante información a este respecto.

[3] Smith A.J. Cache Mernories. ACM Computing Surveys vol 14,3, (1982), 473-530.
[4] Stone H. S. High Performance Computer Architecture Addison-Wesley, 1988.
[5] Przybylski et al. Performance Tradeoffs in Cache Design. Proeeedings of the 15th Annual International Symposium on Cornputcer Architecture, (1988) ,290-298.

En cuanto al algoritmo de búsqueda existen dos alternativas;

Búsqueda por demanda:

La búsqueda por demanda se aplica cuando la memoria cache no es capaz de satisfacer una referencia a memoria, y se trae desde memoria principal el bloque que contiene la palabra de memoria solicitada.

Búsqueda Anticipada:

La búsqueda anticipada consiste en llevar a memoria cache aquellos bloques de memoria que cree se requerirán a futuro.

El principal problema que se presenta en la búsqueda anticipada es un fenómeno conocido como polución de memoria. Es decir, los bloques que se transfieren en forma anticipada *a* la memoria cache pueden contaminar a ésta obligando a desplazar bloques aun útiles que están presentes; este problema se analiza en forma extensa en la bibliografía revisada.[6] [7] Si se considera el uso de búsqueda anticipada hay tres aspectos que deben considerarse: ¿Cuándo iniciar una búsqueda anticipada? y ¿Qué tratamiento dar a los bloques traídos en forma anticipada al momento de reemplazar un bloque en la memoria cache?. Dada la necesidad de que la implementación por hardware sea sencilla y rápida, el único bloque que resulta atractivo de traer en forma anticipada al cache es aquel inmediatamente a continuación del bloque en uso; es decir, si se encuentra en uso el bloque ***i*** sólo el bloque ***i+1*** se considerará atractivo de llevar en forma anticipada a la memoria cache. Esta técnica se conoce como OBL (One Block Lookahead). Respecto a la pregunta de cuando iniciar la búsqueda anticipada de un bloque no existe consenso en la bibliografía sobre el tema, las alternativas a considerar son: siempre, cada vez que se produce un desacierto o cuando se ha logrado cierto grado de avance en las referencias sobre el bloque anterior.

[6] Smith A.J. Sequential Memory prefetcching in Memory Hierarchics. IEEE Computer vol 11,12 (1978), 7-21.

[7] Smith A.J. Sequentiality and prefetching in database systems. ACM trans.Database Systems. vol 3,3 (1978) 223,247.

Otro aspecto a considerar es qué tratamiento dar a un bloque que se ha traído en forma anticipada a la memoria cache y que aún no ha sido usado al momento de elegir que bloque reemplazar frente a un desacierto, en ([3]) se recomienda ponerlo al tope en un algoritmo LRU, es decir, como aquel más recientemente usado. Finalmente, hay que considerar que la búsqueda anticipada (PREFETCH) está implícita en todo algoritmo de manejo de memorias cache, ya que la búsqueda se hace por bloques de modo que siempre se están trayendo en forma anticipada palabras de memoria principal al cache, por lo que si un algoritmo de búsqueda anticipada opera exitosamente, podría significar que el tamaño del bloque ha sido mal escogido, lo que debería llevar a un rediseño de la memoria cache.

Otra alternativa que debe considerarse en el diseño de una memoria cache es la posibilidad de usar una estructura de dos niveles, con un nivel muy rápido y pequeño y un secundo nivel no tan rápido pero de mayor capacidad. Los primeros artículos ([3]) que tratan el tema no encuentran justificable el uso de un cache de dos niveles, pero con la introducción de microprocesadores que incluyen un pequeño cache dentro del mismo chip, el cual es muy rápido, esta alternativa se ha vuelto muy atractiva.

Por último en esta breve revisión de los aspectos de diseño de memorias cache, habría que destacar que la tendencia actual pareciera tender a uso de memorias cache separadas para datos e instrucciones, lo cual permite un mejor manejo de las características de localidad de los programas, y las organizaciones de los datos lo que se traduce en un aumento de la velocidad de operación de los sistemas computacionales.

[3] Smith A.J. Cache Mernories. ACM Computing Surveys vol 14,3, (1982), 473-530.

3. ALGORITMOS DE REEMPLAZO EN MEMORIAS CACHE

El principal objetivo de un algoritmo de reemplazo es retener aquellos bloques que se requerirán en un futuro cercano desplazando aquellos que ya no son útiles y aquellos cuyas próximas referencias se encuentran en el futuro más distante. Prácticamente todos los computadores usan el algoritmo LRU (Least Recently Used) para el manejo de los bloques en la memoria cache; las otras alternativas dignas de ser consideradas son FIFO (First Input First Output) y Random. A continuación se analizará cada una de estas posibilidades.

3.1 Algoritmo Random

Un bloque bi es elegido al azar entre bloques que forman el conjunto en el cual se ha producido un desacierto. Esta política es contraria al principio de localidad por lo cual no es recomendada; sin embargo, algunos resultados de simulaciones indican que al utilizar el algoritmo Random se obtienen razones de acierto superiores a los algoritmos FIFO y LRU.[8] Posee las ventajas de que por un lado su implementación requiere un mínimo de hardware y no es necesario por otro lado almacenar información alguna para cada bloque.

3.2 Algoritmo FIFO

En este caso los bloques dentro del conjunto están ordenados de acuerdo a la secuencia con que son cargados. Cuando se debe reemplazar un bloque se elimina de la memoria cache aquel que fue cargado en primer lugar; los bloques siguientes son removidos en el mismo orden.

El principal problema del algoritmo FIFO se presenta cuando un bloque es requerido repetidamente, para ejemplificar esta situación, consideremos un conjunto formado por cuatro bloques y la siguiente secuencia de bloques que son requeridos sobre ese conjunto:

[8] Smith A.J., Goodman J.R. Instruction Cache Replacement Policies and Organizations. IEEE Transactions on Computers vol C-34 pp 234-241 (march 1985)

a b a d g a f d g a f c a h a

Después de 5 referencias (y 4 desaciertos) los bloques que constituyen el conjunto son (a, b, d, g). La sexta referencia " a " es satisfecha por la memoria cache. En la séptima referencia " f " se produce un desacierto y un bloque del conjunto debe eliminarse; el algoritmo FIFO elimina el bloque " a " (el primero que entró) y deja dentro el bloque " b " que no es usado más, contraviniendo el principio básico de un algoritmo de reemplazo.

Una variación del algoritmo FIFO, conocida como "Clock" o FINUFO (First In Not Used First Out), soluciona el problema de reemplazar un bloque frecuentemente usado y consiste en agregar por un lado un bit de 'uso', que es puesto en '1' cuando un bloque es usado con posterioridad a su carga inicial.

Adicionalmente los bloques dentro del conjunto se ordenan en forma de una cola circular con un puntero que apunta al bloque que será reemplazado. Al producirse un desacierto se examina el bit de uso del bloque apuntado por el puntero. Si está en '0' se reemplaza ese bloque; si está en uno, se borra el bit de uso y se avanza el puntero una posición continuando este procedimiento hasta encontrar un bloque cuyo bit de uso esté en '0'. En el ejemplo antes visto para el algoritmo FIFO luego de la sexta referencia el estado de la cola es:

bloques	a b d g
puntero	**x**
bit de uso	1000

La séptima referencia es sobre el bloque f, por lo tanto hay un desacierto, como el puntero apunta al bloque a que tiene el bit de uso en '1', éste se pone en '0', se avanza el puntero al bloque " b " el cual tiene el bit de uso en '0' por lo cual es reemplazado y el puntero queda apuntando al bloque " d ", por lo tanto el estado de la cola luego de la séptima referencia es:

bloques	a f d g
puntero	**x**
bit de uso	0000

3.3 Algoritmo LRU

El algoritmo más usado es el algoritmo LRU. Tiene la ventaja de que luego de cada referencia, se actualiza una lista que indica cuan reciente fue la última referencia a un bloque determinado. Si se produce un desacierto, se reemplaza aquel bloque cuya última referencia se ha producido en el pasado más lejano, diversas simulaciones indican que las mejores razones de acierto se producen aplicando este algoritmo.

3.4 Situación particular de Análisis

Varios autores analizan la situación particular cuando el programa entra en un lazo cuya longitud es comparada con el tamaño de la memoria cache. Se analizará también aquí esta situación considerando una memoria cache de longitud C y un lazo de longitud L, considerando como responden los algoritmos Random, FIFO y LRU cuando L<C, L=C y L>C. Se usará asignación totalmente asociativo, análisis que puede aplicarse al caso de mapeo asociativo por conjuntos. Cabe destacar que si la memoria cache tiene mapeo directo, el rendimiento de todos los algoritmos de reemplazo será el mismo ya que al momento de reemplazar un bloque solo existe una única posibilidad.

Si L£ C, el rendimiento de los algoritmos FIFO y LRU será el mismo. Se producen L desaciertos cuando el programa entra en el lazo; posteriormente, la razón de acierto será uno, es decir, en estado estacionario la razón de acierto es uno. En ([8]) se plantea que en estado estacionario la razón de acierto para el algoritmo Random también es uno, lo que es cierto. Lo que no se considera en ese artículo es que el algoritmo Random puede tomar un tiempo mayor en alcanzar un estado estable. Podemos asegurar que con los algoritmos FIFO y LRU el estado estable se consigue luego de la primera iteración, pero no es posible asegurar cuantas iteraciones toma el algoritmo Random en hacerse estable, por lo que no puede asegurarse que la razón de acierto para los tres

[8] Smith A.J., Goodman J.R. Instruction Cache Replacement Policies and Organizations. IEEE Transactions on Computers vol C-34 pp 234-241 (march 1985)

algoritmos sea igual cuando L £ C, como se asevera en el artículo mencionado. (El artículo es válido en estado estacionario, no transiente).

Al considerar el caso con L>C, se tendrá que los algoritmos FIFO y LRU, tendrán una razón de acierto igual a cero. Cabe hacer notar que los aciertos y desaciertos que se consideran en este estudio son a nivel de bloques, por lo que el rendimiento a nivel de palabras de memoria será superior. Para ejemplificar esta situación consideremos un cache de 4 bloques A, B, C, D y un lazo de 5 bloques a, b, c, d, e; al iniciar las iteraciones el estado del cache es:

A B C D

w **x** *y* z

donde w, x, y ,z son la ultimas referencias antes de entrar al lazo. Luego de la cuarta referencia y 4 desaciertos, el estado del cache es:

A B C D

a b c d

La quinta referencia es sobre el bloque "e" por cual se produce un desacierto y se remueve el bloque "a", lo que produce un desacierto en la sexta referencia y así sucesivamente. En consecuencia, puede plantearse el siguiente teorema que es una generalización del teorema 1[8].

Teorema: Para una memoria cache de C bloques y un lazo iterativo de L bloques que se repite N veces, la razón global de acierto H, para los algoritmos de reemplazo FIFO y LRU es:

$$H = \frac{L\,(N-1)}{LN}$$

cuando L £ C y H=0 cuando L>C

Este teorema, a diferencia del planteado en la referencia citada, considera el efecto transiente de la primera iteración.

[8] Smith A.J., Goodman J.R. Instruction Cache Replacement Policies and Organizations. IEEE Transactions on Computers vol C-34 pp 234-241 (march 1985)

Para resumir, cuando L£ C, en estado estacionario cualquier algoritmo dará una razón de acierto igual a uno, ya que no es necesario reemplazar ningún bloque. Si L>C, los algoritmos FIFO y LRU dan el peor valor posible para la razón de acierto.

En muchos artículos los autores citan un algoritmo de reemplazo óptimo (llamado OPT en ([4])), el cual reemplazaría el bloque que se requerirá en el futuro más lejano. Obviamente este algoritmo no puede ser implementado en forma práctica dado que requiere conocer las futuras referencias. Apliquemos este algoritmo hipotético al ejemplo planteado; como se ha visto el estado del cache luego de la cuarta referencia es:

A B C D

a b c d

La quinta referencia es sobre el bloque “e”; el algoritmo óptimo reemplazaría aquel que se requiere en el futuro mas lejano, es decir, el bloque “d”. Por lo tanto luego de la quinta referencia el estado es:

A B C D

a b c e

por lo cual se producirán aciertos en las tres próximas referencias. En consecuencia, si L>C se producirán (L-C) desaciertos en cada iteración en estado estacionario y (C-1) aciertos, por lo tanto en estado estacionario se producen (C-1) aciertos seguidos de (L-C) desaciertos, en consecuencia generalizando el teorema 2 de ([8]) se tiene:

Teorema: Para una memoria cache de C bloques y un lazo iterativo de L bloques que se repite N veces, la razón global de acierto óptima es:

$$H = \frac{N(C-1)-L}{NL}$$

cuando L>C

Restaría analizar cómo es la razón de acierto del algoritmo Random cuando L £ C. Intuitivamente, se puede establecer que la razón de acierto será mayor que cero, por lo cual aquí encontraremos un caso particular en que el algoritmo Random nos dará una razón de acierto superior a los algoritmos FIFO y LRU. En ([8]) se plantea una demostración mediante un modelo de Markov en que se obtiene una razón de acierto en estado estacionario.

$$H = \frac{(C-1)}{(C+1)}$$

para el algoritmo Random, cuando L = C + 1. El problema está en conocer cuántas iteraciones demora el algoritmo Random en alcanzar el estado estacionario y que ocurre para un valor pequeño de N, por lo cual, si bien el teorema planteado es correcto, su aplicación es limitada a casos muy particulares y con N grande. Lo que no merece duda es que el rendimiento de algoritmo Random es superior a los algoritmos FIFO y LRU cuando L £ C.

En cuanto a la aplicación del algoritmo FINUFO al caso analizado se puede concluir que su rendimiento es igual al algoritmo FIFO ya que el bit de uso, que lo distingue, no es puesto en uno en ningún momento.

3.5 Solución Planteada

Como se ha planteado para la situación analizada, los algoritmos FIFO, LRU y FINUFO presentan un buen rendimiento cuando L £ C y su rendimiento es pobrísimo cuando L > C. El algoritmo Random que podría parecer atractivo cuando L > C, no es recomendable cuanto es contrario al principio de localidad, principio que es básico al planteamiento de un sistema de memoria jerarquizada. También hay que considerar que si bien se planteó que la memoria cache es transparente para el usuario, los últimos avances en compiladores, hacen pensar que es posible que el compilador entregue ciertas directivas al sistema de manejo

de la memoria cache. Basándose en estas consideraciones, se plantea la siguiente solución.

En la bibliografía consultada, en particular en ([9]), se plantea que en general el orden en que se encuentran los rendimientos de los algoritmos de reemplazo analizados es: LRU, FINUFO, FIFO, Random, por lo tanto la solución que se plantea basada en el algoritmo FINUFO, se justifica ya que por un lado el rendimiento general de FINUFO, es levemente inferior a LRU, en el caso analizado se tiene que ambos algoritmos responderían en igual forma. Se plantea entonces como solución usar el algoritmo FINUFO, al cual habría que agregar un bit adicional que indique cuando se encuentra el programa dentro de un lazo iterativo, este bit debe ser controlado por el compilador, el cual debe marcar con un '1' aquellos bloques que se encuentran dentro de un lazo. Para explicar su operación consideremos la siguiente secuencia de referencias (C=4)

...w x y z a b c d e a b c d e ....d e f g h i**...**

el estado luego de accesar el bloque z es:

	A B C D	
bloques	w x y z	
puntero	x	bit I =0 (global)
bits i	0000	

no se muestra el estado de los bits de uso ya que están siempre en '0', pues con I=1, se descarta su uso.

Luego de 4 referencias y 4 desaciertos el estado de la cola circular es:

	A B C D	
bloques	a b c d	
puntero	x	bit I=0
bits i	1 1 1 1	

La siguiente referencia es sobre el bloque e, por lo tanto hay un desacierto, de acuerdo al puntero debiera reemplazarse el bloque a, como todos los bits i están

[9] Baer Jean Loup

en '1', el bit global se pone en 1 también, lo que produce que el controlador de cache cambie su modo de operación, con I=0 reemplaza y avanza, con I=1 retrocede y reemplaza por lo tanto, la cola circular queda

	A B C D	
bloques	a b c e	
puntero	x	bit I=1
bits i	1111	

las siguientes referencias a, b, c son satisfechas por la memoria cache, por lo que no hay cambios. El siguiente desacierto se produce al accesar el bloque d, luego el controlador pregunta por los bits i que siguen estando todos en '1 ' , luego el bit I sigue en '1', el puntero retrocede y reemplaza el bloque c, que es precisamente el que se requerirá en el futuro más lejano, así, siempre se producirán C-1 aciertos seguidos por L-C desaciertos. Veamos ahora que ocurre al terminar el lazo de iteración, luego del último acceso dentro del lazo habrán en el cache 4 bloques que pertenecen al lazo y uno de ellos será el bloque e, la situación podría ser por ejemplo:

	A B C D	
bloques	a e c d	
puntero	x	bit I=1
bits i	1111	

la siguiente referencia es sobre el bloque f y hay un desacierto, como todos los bits i están en '1', I sigue en '1', el puntero retrocede y reemplaza el bloque a, quedando:

	A B C D	
bloques	a e c d	
puntero	x	bit I=1
bits i	0111	

la siguiente referencia, produce un desacierto, pero ya no están todos los bits i en '1' por lo que el bit global I pasa a '0' y el controlador recupera su modo normal de operación.

El algoritmo planteado tiene un rendimiento igual al algoritmo OPT dentro de un lazo iterativo y se acerca al LRU en casos más generales por lo cual desde el punto de vista de su rendimiento no tiene reparos, es posible sí que su rendimiento se vea afectado cuando el programa sale de un lazo e inmediatamente entra en otro o cuando hay lazos anidados.

Las modificaciones de hardware que se requieren son pocas y no creo que haya dificultad en implementarlas, tal vez lo más negativo es que cada bloque en memoria debe tener el bit i asociado y esto pudiera ser más difícil o bien ineficiente de implementar, el que requiera de directivas entregadas por el compilador no es una novedad ya que existe bibliografía que se refiere al tema, en resumen como todo planteamiento original, debe ser analizado en detalle para analizar todos sus inconvenientes y ver si es posible superarlos.