

Tác dụng của cache

Hiệu quả hệ thống máy tính của bạn rất phụ thuộc vào kiến trúc và cài đặt bộ nhớ truy cập nhanh.

Các CPU hiện đại phải liên tục xử lý lệnh và dữ liệu. Cứ 1 đến 24 tháng, tốc độ của chúng lại tăng lên 2 lần, trong khi đó tốc độ các chip DRAM dùng làm bộ nhớ chính chỉ tăng lên có vài phần trăm năm. Hậu quả là bộ nhớ truy cập nhanh (cache), một loại bộ nhớ đệm giữa bộ nhớ chính và CPU, đóng vai trò quan trọng để nâng cao hiệu quả toàn hệ máy tính. Trong thực tế, nâng cấp cache có khi tác dụng hơn cả việc nâng cấp CPU.

Quy tắc thông thường của công nghệ bộ nhớ hiện đại là kích cỡ càng nhỏ thì làm việc càng nhanh, nhưng tốn kém hơn. Vì lẽ đó, tất cả các máy tính hiện đại đều có hệ thống bộ nhớ ở tổ chức theo kiểu phân cấp: các phần tử nhanh hơn nằm ở trên, các phần tử chậm và rẻ hơn được đặt phía dưới (xem hình 1). Nằm trên cùng là các thanh ghi (32 thanh số nguyên, 32 thanh số dấu chấm động) với tổng dung lượng là 256B. Tiếp theo là bộ nhớ cache: kích cỡ từ 16KB đến 256KB với thời gian truy cập cỡ vài chu trình CPU. Nhiều loại máy có cache nội đặt trong CPU (on-chip) và cả cache ngoại (off-chip) tạo thành 2 mức kế sát nhau.

Khi một trình nào đó làm việc, dữ liệu được di chuyển lên trên và xuống dưới dọc theo cấu trúc phân cấp. Chuyển lên nếu dữ liệu được truy cập và xuống dưới trong trường hợp bị thay đổi dữ liệu mới phải được đưa từ dưới lên. Đơn vị dữ liệu tham gia vào quá trình như vậy là khối (block) và khối trong cache gọi là tuyến (line). Nói chung, dữ liệu trong một mức là tập con của dữ liệu nằm ở mức thấp hơn.

Sở dĩ phải tổ chức bộ nhớ theo kiểu phân cấp như vậy là vì các trình không truy cập bộ nhớ một cách ngẫu nhiên (random). Nếu trình nào đó truy cập một từ bộ nhớ, khả năng cao hơn cả là truy cập đúng từ đó trong tương lai gần. Khả năng truy cập từ kế tiếp cũng khá cao. Hai đặc điểm này được biết dưới nguyên lý định vị trong thời gian (temporal locality) và trong không gian (spatial locality). Theo nguyên lý thời gian, cần phải lưu giữ từ trong bộ nhớ cache một khi nó được đưa vào. Theo nguyên lý không gian, phải nạp vài từ kế tiếp một lúc.

Kiến trúc cache

Như đã nói ở trên, cache được tổ chức thành các tuyến (lines) - các đơn vị bộ nhớ liên tục cùng kích. Vấn đề đầu tiên phải được giải quyết khi thiết kế cache là: nạp khối từ bộ nhớ vào tuyến nào của cache? Theo kiến trúc đơn giản nhất, kiến trúc ánh xạ trực tiếp (DMC - direct mapped cache), địa chỉ bộ nhớ xác định duy nhất tuyến được dùng để nhớ khối. Ta xét hệ thống dùng địa chỉ hóa 2 bit. Nếu kích cỡ của khối là 64B (2 mũ 6), khi đó 6 bit cuối của địa chỉ dùng làm độ lệch (offset) - byte nào trong khối đang được địa chỉ hóa. Nếu cache bao gồm 1024 (2 mũ 10) tuyến cỡ 64B, mười bit kế tiếp của địa chỉ cho biết tuyến mà khối phải được gửi vào. 16 bit đầu tiên của địa chỉ dùng làm số thẻ A (tag) lưu ngay trong tuyến cùng với dữ liệu.

Khi truy nhập bộ nhớ (tức lúc cần nạp dữ liệu) cache dùng các bit giữa của địa chỉ làm chỉ số (index) cho mảng các thẻ. Thẻ tìm thấy sẽ được so sánh với 16 bit đầu tiên của địa chỉ hiện hành. Nếu trùng, dữ liệu với offset tương ứng sẽ được gửi vào CPU. Nếu không trùng, tuyến được thay bởi khối cần thiết từ bộ nhớ chính.

Kiến trúc DMC ưu điểm ở chỗ bạn chỉ cần 1 lần so sánh để truy nhập cache. Nhược điểm: nếu có 2 khối truy nhập liên tiếp cùng ánh xạ vào một tuyến, chúng sẽ đẩy nhau ra khỏi cache.

Kiểu kiến trúc khác là cache kết giao đầy đủ (fully associative cache - FAC): Khối có thể được gửi vào tuyến bất kỳ của cache. Địa chỉ được chia thành các bit cuối dùng làm offset và các bit đầu dùng làm thẻ. Với kiến trúc nào, phải có một cơ chế quyết định tuyến chứa khối dữ liệu tương ứng. Trước hết các khối có thể được đưa vào các tuyến trống, nhưng khi cache đầy, một vài khối phải được chọn để đẩy ra khỏi cache. Tốt nhất, khối LRU (Least recently Used - dùng gần nhất) được thay thế, tuy nhiên theo đúng cách này có thể khá tốn kém, do đó một biến thể của LRU có thể được sử dụng.

Kiến trúc FAC giải quyết được vấn đề xung đột địa chỉ, nhưng với giá của phần cứng bổ sung dùng để so sánh thẻ với tất cả các thẻ đã có trong cache. Giải pháp trung gian giữa DMC và FAC là kiến trúc kết giao tập hợp (SAC - set associative cache).

Theo SAC, các tuyến được chia thành các tập hợp và các bit giữa của địa chỉ xác định tập hợp sẽ chứa khối. Bên trong mỗi tập hợp tuyến, cache được tổ chức theo kiểu FAC. Cache với 2 tuyến cho một tập hợp được gọi là SAC song lộ (Two way set associative cache) và đòi hỏi 2 so sánh cho một truy nhập. Ngoài ưu điểm là số lần so sánh ít hơn so với FAC, kiến trúc SAC cho phép cài đặt LRU dễ dàng hơn. Chẳng hạn, bạn chỉ cần 1 bit để thực thi LRU trong SAC song lộ.

Hiệu quả

Độ trệch cache (miss rate of a cache) là tỉ lệ phần trăm của số lần tham chiếu bộ nhớ không đáp ứng được bởi cache và do đó đòi hỏi phải tìm kiếm trong bộ nhớ chính. Mục đích đầu tiên của nhà thiết kế máy tính (đúng hơn: thiết kế CPU và bộ nhớ - ND) là tối thiểu độ trệch cache (MR). Thông thường, DMC kích cỡ n có cùng độ MR với SAC song lộ kích cỡ $n/2$. Như vậy, khi so sánh các máy, điều quan trọng không là kích cỡ cache mà là độ kết giao (associativity).

Một vấn đề khác là lệnh và dữ liệu được nhớ trong cùng một cache hoặc trong 2 cache khác nhau. Kiến trúc máy với 2 cache lệnh và dữ liệu riêng biệt được gọi là kiến trúc Harvard (xem hình vẽ: Harvard đối nghịch với Princeton). Sử dụng 2 cache có tác dụng loại trừ khả năng giao thoa tham chiếu lệnh và dữ liệu, cho phép chọn và tối ưu riêng rẽ kích cỡ cache, kích cỡ tuyến và độ kết giao. Dữ liệu và lệnh được tìm kiếm dễ dàng hơn trong cùng một thời điểm.

Có hai nhược điểm của kiến trúc Harvard. Nếu chương trình tự sửa bản thân mình bằng cách ghi các lệnh mới, chúng được ghi vào cache dữ liệu. Trước khi chương trình thực hiện

các lệnh này, cả hai cache phải được vét cạn và những sửa đổi phải được ghi vào bộ nhớ chính để cache lệnh nhận được các lệnh mới từ bộ nhớ chính. Nhược điểm thứ 2 là khi một trình đòi hỏi nhiều bộ nhớ cho cache lệnh hơn cache dữ liệu (hoặc ngược lại), việc tách rời i cache không thể làm được điều đó như trong trường hợp cache đơn.

Tính chỉnh cache

Đặc tính cơ bản nhất liên quan đến cache cho một ứng dụng là phần lớn dữ liệu đang xử lý có nằm trong cache hay không và dữ liệu đang nạp có chứa những thông tin không cần thiết không. Ta xét một số cách tổ chức dữ liệu của chương trình cùng với những thông số thiết kế khác nhau của cache.

Tối ưu cache đối nghịch với tối ưu lệnh.

Sai lầm mà người lập trình hay phạm phải là tối ưu chương trình với giả thiết là tất cả các tham chiếu đều tới cache, quên đi một thực tế là khi trệch cache cần mất thời gian và nhiều u lệnh để giải quyết. Một ví dụ: Người lập trình nhớ các giá trị Boolean dưới dạng số nguyên hoặc ký tự thay cho bit. Cách làm này cho phép bạn xử lý dữ liệu chỉ bằng vài lệnh, nhưng số lượng tuyến của cache lại tăng lên, do đó cũng tăng khả năng trệch cache.

Định vị tốt hơn

Khi dữ liệu không có trong cache, việc định vị (locality) trở nên yếu tố quan trọng. Khi trệch cache, dữ liệu được nạp từ bộ nhớ chính vào cache. Các dữ liệu truy nhập cùng nhau nên được đặt gần nhau trong bộ nhớ.

Ví dụ: Nếu ứng dụng của bạn dùng hai đầu mục dữ liệu cho mỗi nhân viên, tốt hơn cả là ghi chúng trong cùng một cấu trúc, thay vì trong 2 mảng khác nhau.

Tối ưu định vị còn tác dụng khi một lượng lớn dữ liệu cần truy nhập không có trong cache. Chẳng hạn, nếu các đầu mục dữ liệu đang truy nhập là các bản ghi cỡ 6B và kích cỡ tuyến là 8B, khi đó có bản ghi sẽ được trải trên 2 tuyến, nghĩa là truy nhập bản ghi phải qua 2 lần trệch. Bằng cách độn phải bản ghi tới 8B để mỗi tuyến chứa được đúng 1 bản ghi, số trệch cache cho mỗi bản ghi được giảm xuống còn 1 (xem hình: Độn bản ghi).

Tranh chấp và độ kết giao

Hai địa chỉ khác nhau chỉ trong phần thể (ví dụ, tại các bit đầu) sẽ ánh xạ vào cùng một tập hợp của cache. Tình huống này gọi là tranh chấp (conflict). Khi các địa chỉ tranh chấp được truy nhập lần nữa và số lần tranh chấp vượt độ kết giao của cache, cũng địa chỉ như vậy sẽ lại tạo ra trệch cache. Trong kiến trúc DMC (còn gọi là SAC lộ) càng dễ xảy ra các tình huống như vậy. Điều này giải thích nguyên nhân SAC song lộ hiệu quả như DMC với kích cỡ cache gấp đôi.

Tranh chấp trở thành vấn đề lớn khi bộ nhớ truy nhập theo bước đều. Có thể đưa ra ví dụ dùng mảng dọc theo một cột nào đó. Chẳng hạn trong ngôn ngữ C, mảng được sắp xếp th

eo hàng: phần tử $A[3,0]$ và $A[3,1]$ nằm cạnh nhau trong bộ nhớ, trong khi $A[3,0]$ và $A[4,0]$ nằm cách nhau cả một hàng. Nếu kích cỡ cache bằng bội số của khoảng cách giữa các phần tử truy nhập liên tiếp, các phần tử này sẽ ánh xạ vào cùng một tập hợp trong cache. Tình hình còn xấu hơn khi khoảng cách này lớn hơn kích cỡ cache và là bội của kích cỡ cache. Khi đó, mỗi phần tử mảng sẽ ánh xạ vào cùng một tập hợp.

Giải pháp tốt nhất là truy nhập dữ liệu dọc theo hàng sao cho các truy nhập đều tới các vị trí bộ nhớ kế tiếp. Nếu không làm được như vậy, cần phải tránh dùng số lượng cột của mảng bằng lũy thừa của 2 (đặc biệt đối với lũy thừa lớn). Để làm được điều đó, hãy độn thêm vào mảng bởi các cột trống.

Ảo đối nghịch với thực

Cho đến giờ, chúng ta mới nói chung về "địa chỉ". Trong thực tế, có 2 loại địa chỉ: ảo (virtual) và thực (physical) kéo theo 2 kiểu cache khác nhau, phụ thuộc vào cách địa chỉ hóa.

Hệ điều hành cho phép thực hiện đa nhiệm nâng cao như Unix và OS/2 tạo ảo giác là mỗi chương trình chạy trên máy riêng của mình. Điều này cho phép mỗi chương trình truy nhập và cấp phát bộ nhớ độc lập với các trình khác mà không phải lo vi phạm vùng bộ nhớ của trình khác.

Trong thực tế, mỗi byte của RAM có địa chỉ riêng. Hệ điều hành và phần cứng cùng hợp tác để tạo ảo giác này bằng cách định nghĩa 2 kiểu địa chỉ ảo và thực (còn gọi là vật lý) và điều khiển việc chuyển đổi giữa hai loại địa chỉ này. Các chương trình dùng địa chỉ ảo, còn bộ quản lý bộ nhớ hệ thống cần đến địa chỉ thực.

Hệ điều hành cấp phát bộ nhớ cho chương trình dưới dạng các trang (page) kích cỡ cố định, thường cỡ 4MB. Hệ còn quản một bảng cho mỗi chương trình mang thông tin về ánh xạ các trang ảo sang các trang thực. Mỗi khi chương trình truy nhập một địa chỉ ảo, hệ thống phải ngó vào bảng này để tìm địa chỉ thực tương ứng.

Tìm kiếm trong bảng như vậy cũng mất khá thời gian. Để nâng cao hiệu quả, các bộ vi xử lý sử dụng cache đặc biệt gọi là TLB (translation look-aside buffer) để ghi nhớ các chuyển đổi địa chỉ hay dùng nhất. Như vậy khi chuyển đổi trang không có trong TLB, hệ điều hành phải ngắt công việc của chương trình và chuyển sang tìm trong bảng chuyển đổi trang, nạp vào TLB bằng lệnh ưu tiên và cuối cùng trả điều khiển về chương trình ban đầu. Trong khi trệch cache tốn không quá 10 chu trình, trệch TLB khá tốn kém: các tuyến phải được vét cạn, các thanh ghi được ghi lại, trình tìm kiếm được khởi động và sau đó lại khôi phục nội dung ban đầu của các thanh ghi.

Quá trình này tốn hàng tá, có khi hàng trăm chu trình. Vì lẽ đó, các TLB thường phải có độ kết giao lớn hơn các loại cache khác.

Tìm kiếm và lưu trữ dữ liệu trong cache có thể dùng địa chỉ ảo và địa chỉ thực. Thiết kế cache thường khó thay đổi, nhưng tùy chọn này cũng ảnh hưởng đến hiệu quả hệ thống. Ta xét 2 kiểu cache khác nhau.

Cache dùng địa chỉ ảo.

Cache loại này có một số ưu điểm. Bộ điều khiển cache không phải chờ biến đổi địa chỉ xong rồi mới bắt đầu tìm địa chỉ trong cache, nghĩa là cache cung cấp dữ liệu nhanh hơn. Cũng vậy, do dùng địa chỉ ảo, vận hành tương tự của chương trình dẫn tới cách dùng cache tương tự.

Điều này không xảy ra đối với cache ánh xạ vật lý, khi mà hệ điều hành có thể cấp phát các trang thực khác cho chương trình trong những lần chạy khác nhau. Kết quả là các thẻ dùng cho các địa chỉ sẽ khác với các thẻ dùng trong lần chạy khác, ngay cả nếu cùng tính toán như vậy được lặp lại. Điều này nghĩa là các lần chạy khác nhau tạo ra những tranh chấp khác nhau và hiệu quả có thể khác biệt đáng kể, đặc biệt là khi dùng cache thực kiểu DMC.

Cache dùng địa chỉ thực.

Cache loại này có 2 ưu điểm lớn. Trước hết, nếu một cache ngoài chip (off-chip) được thiết kế cho CPU với MMU (memory management unit) trong chip, địa chỉ truyền bởi CPU đã được chuyển đổi và cache dùng địa chỉ thực là lựa chọn duy nhất. Kế đó, do tất cả các địa chỉ dùng cho một vùng vật lý duy nhất thay cho vùng với địa chỉ ảo không cố định, dữ liệu có thể được giữ lại trong cache khi hệ điều hành truyền điều khiển từ ứng dụng này sang ứng dụng khác. Trong trường hợp dùng địa chỉ ảo, dữ liệu phải được vét khỏi cache mỗi khi có một điều khiển được truyền. Nói cách khác, ứng dụng A có thể đọc nội dung tại địa chỉ 0 của ứng dụng B, thay cho nội dung theo địa chỉ 0 của chính mình.

Vì lẽ đó, các cache dùng địa chỉ thực thường cho hiệu quả cao hơn trong môi trường đa nhiệm, đa luồng khi đó chuyển từ ứng dụng này sang ứng dụng kia (gọi là context switch) rất hay xảy ra. Cache dùng địa chỉ ảo có thể được sửa sao cho nó có khả năng lưu giữ "thẻ tiến trình" (process tag) cùng với thẻ địa chỉ bên trong cache, mà điều này nghĩa là hệ điều hành phải cấp phát thẻ tiến trình cho các ứng dụng và phải vét cạn (flush) cache tại thời điểm có nhiều ứng dụng chạy hơn số thẻ tiến trình hiện hữu.

Các thao tác phi đồng bộ

Giá phải trả cho một lần trệch cache là vài chu trình (cycles) do tốc độ bộ nhớ không tăng kịp tốc độ CPU, cần phải giảm chi phí này. Kết quả là các bộ xử lý được thiết kế với phần cứng hỗ trợ nhằm giảm tối thiểu giá trệch cache.

Trong thiết kế đơn giản, khi có thông báo về trệch cache, bộ xử lý chờ dữ liệu được nạp xong từ bộ nhớ chính vào cache rồi mới tiếp tục xử lý. Các bộ vi xử lý phức tạp hơn không chờ đợi như vậy mà thực hiện các lệnh kế tiếp trong thời gian nạp cache. Nếu lại xảy ra trệch cache lần nữa trước khi lần trệch trước được giải quyết, bộ xử lý tạm ngừng cho đến khi ít nhất một lần trệch tồn đọng được đáp ứng. Nói chung, có thể có hơn 2 tồn đọng như vậy trước khi bộ xử lý tạm dừng. Các CPU cao cấp hiện hành thường cho phép có từ 1 đến 2 tồn đọng.

Trong cache với kiến trúc đơn giản, khi xảy ra trệch cache, cả một tuyến được dành để chứa giá trị nạp vào, sau đó giá trị này được cung cấp cho bộ xử lý. Điều này đảm bảo là lần trệch kế tiếp tại tuyến như vậy sẽ không xảy ra, khi tuyến đó đang trong quá trình truyền.

Cách tiếp cận phức tạp hơn có khả năng nạp cache bắt đầu tại từ cần đáp ứng và quay lại đầu tuyến để tìm các từ khác. Từ nhận được sẽ đưa vào CPU ngay sau khi đến từ bộ nhớ trống và phần còn lại của dữ liệu được truyền khi CPU tiếp tục xử lý.

Vấn đề gọi là kẹt cache (cache jamming) có thể được giải quyết bằng cách đưa một biến trống vào giữa các mảng.

Kết luận

Có nhiều thông số của cache ảnh hưởng tới hiệu quả hệ thống: kích cỡ cache, độ kết giao, kích cỡ tuyến, ảo hay thực, mức phi đồng bộ (tức là số trệch cache tồn đọng). Trong khi bạn không thể điều khiển được các thông số này, hãy để ý đến chúng khi chọn hoặc đánh giá một hệ máy. Kết quả kiểm nghiệm cho thấy đối với một CPU cụ thể, thay đổi cấu hình cache cũng có ảnh hưởng đến toàn hệ thống. Khi các thông số khác nhau, độ kết giao cao hơn nghĩa là hệ máy tốt hơn. Cache kiểu DMC đặc biệt nhạy cảm với vấn đề tranh chấp. Cache dùng địa chỉ ảo đảm bảo hiệu quả đồng đều nhất, nhưng cache dùng địa chỉ thực lại tốt hơn trong môi trường đa nhiệm và đa luồng.

Đối với nhà lập trình, hiểu biết về cache sẽ giúp tránh được các bẫy có khả năng làm giảm đáng kể hiệu quả của chương trình. Nếu bạn thiết kế hoặc chỉnh một ứng dụng dùng tích hợp CPU, hãy tìm cách tối đa khả năng định vị và tránh các chuỗi truy nhập bộ nhớ với địa chỉ tăng theo lũy thừa lớn của 2.

Tô Tuấn
Byte 8/1994