Caches

Luke Skywalker

18 de novembro de 2015

Considere uma cache com mapeamento direto e endereços de memória de x bits organizados da forma abaixo, responda qual o espaço ocupado pela cache no processador em bits. Considere que c bits de controle são usado por linha de cache.

TAG	ÍNDICE	OFFSET
t bits	i bits	f bits

$2^i \times (\underline{(x - i - f)} + c + (2^f \times 8))\ bits$

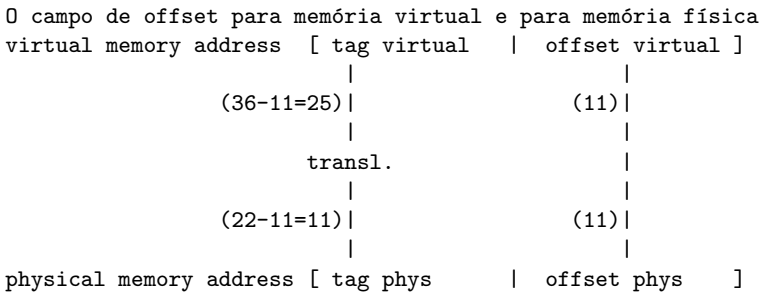
x: igual ao tamanho de bits do endereço, nesse caso 32 bits.
c: igual número de bits de controle por linha de cache.

$2^i \times (\underline{(x - i - f)} + c + 2^f)\ bytes$

Considere um sistema com as seguintes configurações:

1. Memória virtual de 2^{36} bytes
2. Memória física de 2^{22} bytes
3. Páginas de 2^{11} bytes
4. 2 bits extra para o controle das páginas

Informe o tamanho da tabela de páginas em bits.



$2^{36-11} \times (2 + (22 - 11))$

$2^{25} \times (2 + (22 - 11))$

Considere uma cache com mapeamento 4-associativo e endereços de memória de 32 bits organizados da forma abaixo, responda quantas linhas tem a cache.

TAG	ÍNDICE	OFFSET
31-26	25-10	9-0
6 bits	16 bits	10 bits

n-way $\times$ $2^{\text{índice bits size}}$

$4 \times 2^{16} = 262144$ linhas

Considere um sistema com as seguintes configurações:

1. 2^{28} bytes endereçáveis de memória
2. Cache com 2^5 blocos de 2^7 bytes cada
3. Linhas de cache com 1 bit de validade

Qual seria o tamanho efetivo da cache em bits caso ela fosse implementada com um mapeamento 4-associativo?

n-way $\times \frac{2^5}{4} \times ((28 - 7 - 3) + 1 + 2^7 \times 8)$

$4 \times \frac{2^5}{4} \times (18 + 1 + 2^7 \times 8)$

Considere um sistema com as seguintes configurações:

1. Memória virtual de 2^{33} bytes
2. Memória física de 2^{24} bytes
3. Páginas de 2^{11} bytes
4. 6 bits extra para o controle das páginas

Informe o tamanho da tabela de páginas em bits.

$2^{33-11} \times (6 + (24 - 11))$

$2^{22} \times (6 + 13))$

1 Teoria

pg. 452

- Localidade Temporal (localidade no tempo): Se um item é referenciado, ele tenderá a ser referenciado novamente, em breve.
- Localidade Espacial (localidade no espaço): Se um item é referenciado, itens cujos endereços estão perto, tenderão a serem referenciados também, em breve.

  Programas exibem costumeiramente ambos: localidade temporal, a tendência em reusar itens recentemente acessados, e localidade espacial, a tendência em referenciar dados que estão perto daquele dado recentemente acessado. Hierarquia de memória usa a vantagem de localidade temporal deixando itens recentemente acessados mais perto do processador. Hierarquia de memória tiram vantagem de localidade espacial movendo blocos de dados consistindo de múltiplas palavras contíguas para níveis de memória mais acima. Um dado da hierarquia não pode estár em um nível n se não estiver em um nível $n+1$, funciona como uma hierarquia normal.

pg. 454

- Bloco (ou Linha): A unidade mínima de informação que pode estár presente ou não na cache.
- Taxa de Acertos: A fração de acessos achados na memória em um nível da hierarquia de memória.
- Taxa de Erros: A fração de acessos não-achados na memória em um nível da hierarquia de memória. $miss\ rate = 1 - hit\ rate$
- Penalidade por faltas: O tempo requerido para buscar um bloco em um nível mais abaixo na hierarquia de memória, incluindo o tempo de acesso ao bloco, transmita isso de uma nível para outro, insira no nível que experienciou o miss, e então passe o bloco para o requisitador.
- Tempo de Acerto: O tempo requerido para acessar um nível de memória, incluindo o tempo necessário para determinar quando é acesso é um *hit* ou um *miss*[a].

pg. 457 Cache de mapeamento direto é uma cache onde cada localização na memória é mapeada para exatamente uma localização na cache. $X_1, X_2, X_3, \ldots, X_{n-1}$, então o processador requisita uma palavra X_n da cache e se essa palavra não estiver lá, acarretará em um miss, porém o dado referenciado pelo endereço X_n vai ser inserido na cache, na posição segundo a fórmula: $(Block\ Address)\ modulo\ (Number\ of\ blocks\ in\ the\ cache)$[b].

[a] Explorar hierarquia de memória é chave para obter boa performance.

[b] Leituras na memória são mais simples de se tratar do que escritas na memória

pg. 458

- Tag: Um campo em uma tabela, usado para a hierarquia de memória, que contém a informação do endereço requerida para identificar quando o bloco associado na hierarquia corresponde à palavra requisitada.
- Bit de Validade: É um campo na tabela de uma hierarquia de memória, que indica que o bloco associado na hierarquia contém um dado válido (1) ou inválido (0).

459-63 Eu achei muito complicado o jeito que o Patterson criou uma formula para calcular caches diretamente mapeadas, segue a minha solução para calcular caches diretamente mapeadas:

TAG	ÍNDICE	OFFSET
t bits	i bits	o bits

$2^i \times (t + c + 2^o \times 8)$ Essa fórmula mais simples calcula o tamanho total da cache. Sendo que 'c' é bits de controle, por exemplo c=1 (bit de validade). Caso você não tenha os valores diretos de tag, índice, offset. Provavelmente um problema te dará $p\ KB$ (por exemplo, 4 KB) de dados na cache, e dá também quanto cada bloco dessa cache tem: $s\ B$, ou $s\ KB$ (por exemplo, 64 bytes, ou 1 KB). Assim sabendo que é diretamente mapeada, basta dividir $\frac{p\ KB}{s\ B} = Nro.\ Blocos$, ou $\frac{p\ KB}{s\ KB} = Nro.\ Blocos$ a partir do número de blocos, basta você tirar $log_2\ (nro.\ blocos) = indice\ bits$; por exemplo $log_2\ (\frac{2KB \times 2^{10} = 2048\ bytes}{64\ bytes} = 32) = 5$, pois $2^5 = 32$, ou seja: 5 bits mapeiam 32 blocos na cache diretamente mapeada. Sabendo que cada bloco tem 64 bytes é porquê nós temos um offset de $log_2\ (64) = 6$ bits, então temos que $tag = 32 - indice - offset = 32 - 5 - 6 = 21$ bits[c].

TAG	ÍNDICE	OFFSET
21 bits	5 bits	6 bits

464 Em uma cache diretamente mapeada com 64 blocos e 16 bytes por bloco. E no exemplo um endereço 1200_{10} chega para essa cache, onde o dado contido nesse endereço $M[1200_{10}]$ vai para na cache? Simples: $(1200\ //16)\ mod\ 64 = 11$[d]. A operação modular do ponto de vista binário você está pegando do valor binário o que lhe interessa, pois 1200 em binário é 010010110000_2 fazer "// 16" você retira os 4 bits menos significativos "do offset", fazendo agora "mod 64" acha o local onde na cache, pois você pega os 6 bits menos significativos que é o resto da divisão.

470

471

472

473

474

475

476

477

478

479

480

[c] Confira os resultados na aplicação: http://cachesapp.herokuapp.com/.

[d] // significa divisão inteira, nessa divisão inteira você está na verdade retirando os bits menos significativos, é como se estivesse fazendo deslocamento.

481
482
483
484
485
486