Exercício Programa 2 – Simulador de caches – 2020.Q1 - UFABC

Henrique de Abreu Piccolo RA11113108

Introdução

A hierarquia de memórias é algo fundamental na arquitetura de computadores, otimizando a performance do hardware buscando um custo compatível com o projeto. Em grosso modo a hierarquia de memórias pode ser comparada aos “espaços” para ferramentas de um mecânico sendo o armário o local de acesso mais lento porém de maior espaço de armazenamento disponível, a caixa de ferramentas um local de velocidade de acesso e espaço intermediários e os bolsos sendo o local de acesso rápido porém com menor capacidade de armazenamento. Em um computador esses armazenamentos podem ser comparados o armário com HD, local com maior espaço de armazenamento porém muito mais lento para acessar, a memória RAM com a caixa de ferramentas e a memória cache com os bolsos, de certa forma as mãos do mecânico ainda seriam equivalentes aos registradores do processador e sempre é desejável que a ferramenta/dado esteja no local de acesso mais rápido.

A execução de uma tarefa por um processador depende de um equilíbrio parecido com a com a execução da tarefa pelo mecânico, por melhor e mais rápido que ele seja se durante uma tarefa ele tenha que parar a tarefa para buscar a ferramenta na caixa de ferramentas isso tem um impacto no tempo de execução. Se esses acessos, forem muito frequentes pode chegar ao ponto do tamanho da caixa de ferramentas/memória RAM ou bolsos/cache impactar mais no tempo de execução do que a habilidade do mecânico/velocidade do processador.

Em números aproximados uma memória cache demora entre 0.5ns – 2.5ns para acessar um dado e tem um custo entre $500 – $1000 / GB. A memoria RAM tem o tempo entre 50ns – 70ns e custo entre $10 – $20 / GB e um HD tradicional tem o tempo entre 5.000.000ns – 20.000.000ns e custo entre $0.05 – $0.10 / GB. Assim podemos ver que é inviável em termos de custo guardar todos os dados apenas na memoria cache ou mesmo na RAM, e em termos de performance é inviável apenas o HD por isso um equilíbrio entre os níveis é fundamental para que um dispositivo tenha um custo viável e um desempenho adequado. Nesse simulador iremos abordar um simulador onde pode se testar o comportamento da cache.

A memória cache

A memória cache pode ser dividida e uma série de blocos de bytes e a soma de todos os blocos define o espaço total da cache. Em um cenário ideal qualquer endereço de memória poderia ser armazenado em qualquer bloco porém para fazer isso exigiria uma controladora mais sofisticada e mais cara porem há como distribuir essa associatividade de blocos a endereços de memória de uma maneira eficiente. Essa relação de qual endereço pode ser associado a quantos blocos é chamado de associatividade de cache que varia do mais restritivo, chamado de mapeamento direto, um formato intermediário, chamado de n-vias que é o mais usado, ou o menos restritivo chamado mapeamento totalmente associativo.

Associatividade de cache

É um formato onde cada fica associado a um conjunto de endereços, como distribuir esses conjuntos depende da quantia de blocos e da quantia de bytes em cada bloco. Aparte do endereço de memória que endereça os bytes do blocos é chamada de offset e tem o tamanho em bits definido por $\log_2(bytes\ por\ bloco)$. A parte do endereço que determina qual conjunto de endereços pode ser alocado nesse bloco é chamada de set e é definida pelos bits significativos após os bits de offset, a quantia de bits varia da forma $\log_2(quantia\ de\ blocos)$. Os bits restantes que compõem o endereço formam o que é chamado de TAG. Assim em uma busca na cache o offset indica o byte em que o endereço inicia, o set em qual bloco o endereço deve ser buscado e se a TAG for a mesma que estava na cache então o endereço buscado é o mesmo que estava na cache, quando isso o corre é chamado de hit, quando o endereço não está na cache ocorre um miss. O offset na busca acaba não tendo influencia pois a cache acaba acessando todos os bytes do bloco.

Então quando o processador quer ler um endereço que está na cache temos um hit, quando não está chamamos de miss e tera que ser acessado uma memória superior com mais espaço mas mais lenta,no caso a RAM, e o endereço que estava no bloco é substituído. Quando a cache permite guardar mais de um endereço associado a um SET chamamos de cache associativa, cada conjunto extra de endereços que podem ser armazenados é chamado via. Então uma cache que permite guardar 2 endereços por conjunto é uma cache de duas vias. Quando a cache tem um único conjunto com todos endereços e eles podem assumir qualquer posição na cache ela é chamada de totalmente associativa.

Escritas na cache e na RAM

Foi comentado sobre leitura em casos de hit e miss, com a escrita a cache pode lidar de maneiras diferentes. Quando o endereço solicitado está na cache temos um hit e o processador escreve na cache os dados desse endereço. Em casso de miss o processador pode escrever diretamente na memoria RAM, ou escrever na cache para então escrever na RAM. O primeiro formato é chamado “no-write alocate” o segundo formato é chamado “write alocate”. No simulador foi implementado apenas a política “write alocate”.

Também há duas outras maneiras de lidar com a escrita. Uma vez que a ocorre uma escrita na cache ela pode atualizar instantaneamente na RAM, chamada write through (WT), ou esperar apenas quando o bloco for ser substituído e então escreve na RAM, essa política é chamada write-back (WB). A vantagem do WT é manter a cache e RAM atualizadas e ser mais simples de implementar, normalmente essa política é combinada com o formato no-write alocate. A vantagem do WB é em casos de seguidas escritas com hits na cache ela evita escritas na RAM, que consomem mais tempo, e com poucos riscos, perca de dados ocorre apenas casos extremos como queda de energia antes do processador salvar os endereços desatualizados da cache na RAM.

O simulador trabalha com as duas políticas em paralelo mas utilizando o formato write alocate. Em casos de escritas ele indica que em WT ele iria escrever na memória RAM e deixa sinalizado que em WB o endereço estaria desatualizado na RAM, chamado de “Dirt”. Quando um bloco esta desatualizado e vai ser substituído o simulador conta que em WB ocorre uma escrita na RAM, no final da simulação as contagens em WT e WB, assim como a quantia de hit, miss, acessos a leitura e escrita são fornecidos.

Politicas de substituição

Como foi mostrado em associatividade de cache pode armazenar mais de um endereço para um conjunto de endereços. Nesses casos após todas as vias serem ocupadas há mais de uma maneira de escolher a via que vai ser substituída e isso pode influenciar no desempenho da cache. Neste simulador foram implementados 4 politicas de substituição:

- Aleatória, “Random”, a via a ser substituída é escolhida aleatoriamente
- LRU, “Least Recently Used”, a via substituída é a via usada menos recentemente
- MRU, “More Recently Used”, a via substituída é a via usada mais recentemente
- FIFO, “First In First Out”, as vias são substituídas sequencialmente, a primeira a ser escrita é a primeira a ser substituída.

Caches multiníveis

A cache pode ser composta por caches de níveis intermediários, L2 não sendo comum caches maiores que o nível L3. A ideia das caches de níveis superiores é ser um intermediário antes da RAM. Elas são mais rápidas que a a RAM mas não tão rápidas quanto a L1 mas são maiores que a L1. Isso ajuda a equilibrar o custo benefício que foi comentado inicialmente.

Algo muito comum também é ter um nível separado de leitura e escrita da memória RAM, um nível apenas para as instruções do processador. Esse nível quando existe é paralelo a L1, muito rápido e muito pequeno.

Os simulador permite uma cache de instruções e dados separados, assim como níveis superiores, L2, L3… Porém para níveis superiores pode ocorrer alguns erros na contagem de misses e hits devido a forma que foi implementada a politica write back. Também ocorrerão erros se o tamanho de bytes por bloco da cache L1 for maior que oda L2, mas isso não seria usual. Ainda com essas limitações assim essa simulação é útil como um referencial.

Assimulações para cache L1 unificada ou não com cache de instruções LI ocorrem normalmente.

Implementação

Cada nível de cache é formado pelo método cachecreate(nivel,Ilinhas,Ibytes,Vias, Politicasub) , ele recebe o nível, índice de quantas linhas a cache tem, índice de quantas vias de associatividade a cache vai ter e a política de substituição. O índice de linhas e índice de bytes são respectivamente a quantia de bits para set e quantia de bits de offset. O método monta uma tabela onde cada linha é formada por:

- índice de inicialização: utilizado para no inicio da cache indicar quais vias foram preenchidas e estão válidas, no para a política de substituição “Fifo” continua sendo utilizado para manter a ordem da fila.
- Dirt bit para a via: utilizado para simulação do write back.
- TAG para avia: espaço para armazenar a TAG associada ao endereço salvo na cache naquela via.

Conforme a quantia de vias cache tiver são adicionados novamente as colunas do dirt bit e TAG até completar a quantia de vias. Caso a política de substituição seja LRU ou MRU ainda é adicionado n colunas com n=número de vias da cache, para manter a ordem de qual foi a via utilizada mais ou menos recentemente. As informações da cache como quantia de vias e bytes por bloco são salvas em um dataframe infcache.

O trace com as informações da instrução, endereço e o tamanho em bytes é importado como um data frame, é importante a primeira linha do arquivo csv ser ‘operacao,endereco,tamanho’ para a correta interpretação de cada informação. Os endereços serão convertidos para inteiros em base 10, para isso é importante informar a base em que os endereços do trace foram gerados. Para simplificar a geração de relatórios as instruções que escrevem na memória serão tratadas da mesma forma então instruções ‘M’ serão passadas como ‘S’.

Então após declarados o formato das caches para simulação e carregado o trace o simulador faz o endereçamento de acordo com os bits de offset e set para cada cache. Em casos em que é necessário acessar mais de um bloco a quantia de blocos necessária é calculada e é realizado um laço, assim após cada acesso a cache o endereço é atualizado com a soma da quantia de bytes por bloco até que todos blocos necessários sejam lidos. Caso a cache tenha níveis superiores no case de um miss é acessado o nível superior.

O método acessacache(Offset,Set,Tag,instrucao,endereco,nBytes,cache,Nivel,Vias,Politica sub) é o método principal o onde de fato a cache é acessada verifica se o endereço estava na cache e em aplica a política de substituição e checagem para WT e WB. Nesse método os termos stallWB e stallWT são usados de maneira incorreta, eles apenas verificam se houve uma escrita na memória de acordo com cada política mas não é possível afirmar se haveria um stall, para isso seria necessário implementar um write bufer. Após o acesso todos os dados como: endereço, offset, set, tag, instrução, nível da cache, vias, politica de substituição e resultado são guardados na tabela resultados.

Foi implementado um modo chamado iterativo, se essa variável é setada em 1 o simulador gera uma saída txt com o passo a passo do acesso a cache, a composição do endereço se foi hit ou miss para cada acesso a cache. Há um verificador de segurança que caso a trace seja maior que 40000 linhas ele não gera o arquivo de saída pois essa saída pode facilmente ocupar gigabytes de espaço.

Ao final da simulação a tabelar resultados é utilizada no método CSV, ele transforma a tabela em um data frame utilizando o pandas assim permitindo a coleta de diversas informações. Foi implementado um relatório em csv após cada simulação contendo o total de memória simulado nas caches, informações sobre total de acessos, total de hits, total de misses e percentual de hits. Essas informações são dadas tanto no geral quanto por nível. Também é gerada uma tabela com a quantia de acessos por set para cada cache com o total de hits total de misses e percentual de hits, tanto no geral quanto para escrita e para leitura. O simulador foi pensado para permitir uma possível expansão gerando um relatório mais completo podendo conter gráficos ou outros comparativos. Os relatórios são gerados com o nome sim_”nomedatrace”.csv, e adicionam um numero inteiro caso já exista um arquivo com esse nome. A saída do modo iterativo tem o nome Saida.txt e sempre sobrescreve o anterior.

Resultados

Foi simulado 3 arquivos de trace fornecidos, hello, ls e call, com uma cache de 32kb instruções e 32kb dados, essa cache foi pensada para simular o processador AMD K6-2 de 1998. Em todos os traces os resultados foram bons com a cache de instruções acima de 99,5% de acertos, a cache tinha duas vias e 64 bytes por bloco. O percentual de acertos foi maior nas traces maiores o que leva a crer que mesmo 32kb é o suficiente para armazenar a maioria das instruções mais utilizadas nos testes. Apesar da alta taxa de acertos é importante lembrar que essas traces, hello, ls e call, mesmo em 1998 seriam tarefas simples. Mesmo avaliando a cache de dados os resultados foram próximos de 95%.

Utilizando o trace hello foi feita uma série de novas simulações com diversas configurações de cache, mantendo 32kb para cada nível. Os melhores resultados foram com LI e L1 tendo 2048 e 1024 bytes por bloco em mapeamento totalmente associativo. Apesar que com 4 vias LI tem um desempenho similar ao totalmente associativo e em 4 vias com 512 bytes por bloco L1 já tem um desempenho de 98,55% próximo ao de 98,84 que foi conseguido com totalmente associativo. Todas simulações utilizaram a politica de substituição LRU.

Outro ponto importante é a diferença de escritas entre WB e WT, como ocorrem poucos misses poucos blocos precisam ser substituídos o que reduz bastante escritas na RAM. A diferença na quantia de acessos apesar da trace ser a mesma ocorre pois com blocos menores é maior a necessidade de acessar mais de um bloco para acessar os blocos solicitados pela trace .

Linhas	128	64	32	16	32	16	8	4	1	1	1	1	LI=4 L1=16	1
Vias	2	2	2	2	4	4	4	4	128	64	32	16	4	LI=16 L1=32
Bytes por bloco	256	512	1024	2048	256	512	1024	2048	256	512	1024	2048	LI=2048 L1=512	LI=2048 L1=1024
Memoria total	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536
Total de acessos	268722	266052	264142	264061	266052	264142	264061	263951	266052	264142	264061	263951	263952	263951
Hits	266219	264402	262830	262645	264530	263013	262922	262079	264573	263135	263149	262009	262886	263065
Mises	2503	1650	1312	1416	1522	1129	1139	1872	1479	1007	912	1942	1066	886
Percentual de acerto	99,07	99,38	99,5	99,46	99,43	99,57	99,57	99,29	99,44	99,62	99,65	99,26	99,6	99,66
Escritas em WB	370	234	160	136	218	144	125	208	214	134	98	175	144	98
Escritas em WT	12817	12817	12815	12815	12817	12815	12815	12815	12817	12815	12815	12815	12815	12815
Acessos leitura	46313	46312	46312	46311	46312	46312	46311	46311	46312	46312	46311	46311	46312	46311
Hits leitura	44812	45294	45434	45242	45377	45576	45504	44756	45409	45677	45698	44707	45576	45698
Miss leitura	1501	1018	878	1069	935	736	807	1555	903	635	613	1604	736	613
Percentual de acerto leitura	96,76	97,8	98,1	97,69	97,98	98,41	98,26	96,64	98,05	98,63	98,68	96,54	98,41	98,68
Acessos escrita	12817	12817	12815	12815	12817	12815	12815	12815	12817	12815	12815	12815	12815	12815
Hits escrita	12461	12606	12682	12717	12615	12692	12721	12705	12617	12697	12740	12675	12692	12740
Miss escrita	356	211	133	98	202	123	94	110	200	118	75	140	123	75
Percentual de acerto escrita	97,22	98,35	98,96	99,24	98,42	99,04	99,27	99,14	98,44	99,08	99,41	98,91	99,04	99,41
Percentual de acerto LI	99,69	99,8	99,85	99,88	99,81	99,87	99,88	99,9	99,82	99,88	99,89	99,9	99,9	99,9
Percentual de acerto L1	96,86	97,92	98,29	98,03	98,08	98,55	98,48	97,18	98,13	98,73	98,84	97,05	98,55	98,84

Tabela 1: comparativo para diversas caches utilizando o trace hello