**INSTITUTO TECNOLÓGICO DE AERONÁUTICA - ITA**

DIVISÃO DE CIÊNCIA DA COMPUTAÇÃO

**CES-25 – Arquiteturas para Alto Desempenho**

****

**Projeto Exame**

**Simulador de Sistema de Memória**

**Com dois níveis de cache**

**ALUNO**

Felipe Tuyama de Faria Barbosa ftuyama@gmail.com

**PROFESSORA**

Paulo André Lima de Castro pauloac@ita.br

São José dos Campos, 30 de Junho de 2016

**1. INTRODUÇÃO**

O objetivo deste projeto é implementar um simulador de sistema de memória com dois níveis de cache. Através deste, será possível ao desenvolvedor familiarizar e comparar diferentes configurações de arquitetura (quantitativamente), explorando as soluções vistas em aula para as propriedades principais das caches. Dentre elas, seu tamanho físico, estratégias e políticas de gravação e substituição de blocos.

O simulador foi implementado na linguagem de alto nível Python, uma vez que não é exigido alto desempenho para o programa, neste contexto. No contexto industrial, no entanto, seria recomendável um desenvolvimento mais eficiente, uma vez que a execução de múltiplos benchmarks pode ser demorada.

**2. DESCRIÇÃO**

Da definição de simulador: *“Para ser realizada uma simulação, é necessário construir um modelo computacional que corresponda à situação real que se deseja simular”[1]*, temos a palavra-chave “modelo computacional”.

Ou seja, é preciso descrever os comportamentos que se assemelham à situação real e também adotar premissas que representem o fenômeno a ser simulado. No caso deste simulador, temos o código fonte que descreve o comportamento do sistema de memória e também uma tabela de tempo gasto com cada operação, que representa as premissas adotadas para o sistema.

**2.1 – Descrição do tempo gasto**

Assim, preenchendo as Tabelas de tempos necessários para a realização de cada uma das operações possíveis, temos o seguinte cenário de premissas adotadas pelo simulador:

Tabela 2. Operações causadas por acesso ao Sistema de Memória.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Operação causada por | |  | Situação | |  |
| acesso ao Sist.Mem. | | Acerto em L1 | Acerto em L2 | | Acerto na Memória |
| Leitura | | 1+2=3 |  |  |  |
| Leitura | | Não(\*A) | 2+4=6 | |  |
| Leitura | | Não(\*B) | Não(\*C) | | 60 |
| Escrita | | 2+4=6 (WT) |  |  |  |
| Escrita | | Não(\*A) | 2+4=6 (WB) | |  |
| Escrita | | Não(\*B) | Não(\*C) | | 60 |
| Tabela 3. Operações causadas por subida de de bloco | | | | | |
| Operação causada |  |  | Situação | |  |
| por subida de bloco |  |  |  |  |  |
|  | Descrição |  |  | Tempo Adicional |
| (\*ID) |  |  |  |
|  |  |  |  |  |
| (\*A) |  | Atualização apenas em L1; L2 entrega | |  | 0 |
|  |  | dados a CPU e a L1 | |  |  |
| (\*B) |  | Atualização em L1, memória entrega dados a CPU, L1 e L2. | |  | 0 |
| (\*C) |  | Política de substituição WB encadeia escrita em L2, se necessário (WB). | | 60 (para bloco sujo) - WB  0 (para bloco limpo) | |

(\*ID) Subida de bloco determinada por situação indicada na Tabela 2.

Na situação (\*B)/(\*C), temos uma leitura na memória principal com falhas nas caches L1 e L2. Para a cache L1, basta realizar a substituição sem preocupações adicionais, graças à política de gravação Write Through. Porém, para a cache L2, deve-se checar o bit M (o qual indica bloco sujo, incoerente com o bloco na memória principal). Caso positivo, deve-se realizar o Write Back, que custará o tempo de acesso de escrita à memória principal, além da leitura a ser realizada originalmente.

Porém, na situação (\*D)/(\*E),

**2.2 Descrição do espaço gasto**

Uma observação importante é o sobre o tamanho *real* disponível para dados. Para o desenvolvimento do simulador, foi considerado o tamanho declarado no roteiro para cada uma das caches.

No entanto, foi necessário incluir um bit M (modified) adicional para cada bloco de L2 (4096KB/64bytes = 64 blocos -> 64 bits = 8 bytes), a fim de determinar se aquele bloco sofreu modificações desde que foi trazido da memória principal, sendo assim necessária a operação de Write Back quando ocorrer sua substituição.

Seria possível também adicionar um bit de validade aos blocos (útil em multiprocessadores e na inicialização da cache, marcando posições de memória “vagas”). No entanto esse bit seria utilizado somente no começo do sistema, gastando espaço adicional e ainda acrescentando ineficiência (busca constante de posições livres “inválidas’). Assim, a implementação do LRU utilizando FIFO (First In, First Out) resolve bem esse quesito, podendo dispensar a utilização deste bit.

A implementação de FIFO é outro aspecto a se considerar. A melhor solução em espaço é usar as posições da cache como uma fila circular, mantendo em memória a última posição de bloco alocado (de forma que a posição seguinte será sempre a do bloco mais antigo, facilitando a substituição). Essa estratégia usará espaço adicional da ordem O(log n), em que n é o número de blocos da cache em questão (6 bits para cache L2 e 4 bits para a cache L1).

No simulador existem variáveis adicionais, como “Política de gravação”, “Política de gravação na falha”, “tempo de acesso”, “tempo de tag”, “memória inferior” e “status de hit”, que apenas usam o espaço em memória no simulador, não se aplicando a este quesito de espaço despendido com o sistema de memória em si.

**2.3 Otimização sugerida**

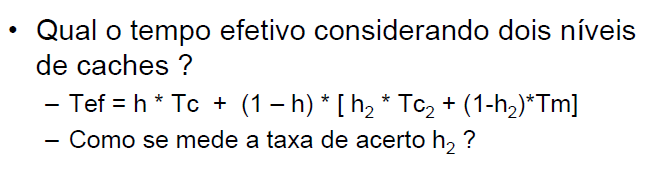
Logo de cara, a solução de implementar LRU usando FIFO me deixou um pouco intrigado por existirem técnicas mais eficientes de se proceder, como o algoritmo do envelhecimento (usando contador) ou mesmo uma pilha dos os blocos da cache.

**3. RESULTADOS**

**3.1 Critérios para resultados & avaliação**

Como medida de desempenho do sistema, temos as estatísticas de acerto e de sucesso do acesso a cada uma das caches. Atribui um significado diferente a cada um destes: a taxa de acerto é a razão entre o número de hits (de uma dada cache/memória) e o número total de todas operações realizadas. A taxa de sucesso é a razão entre o número de hits (de uma dada cache/memória) e o número total de acessos (tentativas com sucesso ou com falha) àquela mesma dada cache/memória.

Temos também o tempo de memória, que indica o tempo total de execução do simulador com base no benchmark fornecido. É um número alto que isolado aparentemente não tem muito sentido. Mas pode ser muito útil para comparar com a melhoria sugerida e também para o cálculo do tempo efetivo do sistema, que abordo no próximo parágrafo.

Outro parâmetro para a medição de desempenho que também foi calculado é o tempo efetivo do sistema de memória (valor intermediário dos tempos de cache e memória), usando a seguinte fórmula para dois níveis de cache [2]:  Respondendo à pergunta do enunciado, a taxa de acerto h2 é dada pela razão entre o número total de hits na cache L2 e o número de acessos (requisições, sendo acertos e falhas) à cache L2. Ou seja, dado que L1 falhou, a taxa de sucesso na cache L2.

**3.2 Resultados para simulação original**

Respondendo à pergunta do enunciado, a taxa de acerto h2 é dada pela razão entre o número total de hits na cache L2 e o número de acessos (requisições, sendo acertos e falhas) à cache L2. Ou seja, dado que L1 falhou, a taxa de sucesso na cache L2.

**3.3 Resultados para simulação otimizada**

Respondendo à pergunta do enunciado, a taxa de acerto h2 é dada pela razão entre o número total de hits na cache L2 e o número de acessos (requisições, sendo acertos e falhas) à cache L2. Ou seja, dado que L1 falhou, a taxa de sucesso na cache L2.

**4. ANÁLISE**

Comparando os resultados obtidos para cada modelo de sistema de memória simulado, observamos que

**5. CONCLUSÃO**

Este projeto permitiu aprender na prática o funcionamento de um sistema de memória e as estratégias adotadas para otimizar sua performance. Particularmente achei bem interessante aplicar o simulador para um arquivo benchmark relativamente grande, podendo obter e visualizar resultados quantitativos da aplicação de uma nova política ou otimização proposta.

**6. REFERÊNCIAS BIBLIOGRÁFICAS**

[1] Wikipedia : <https://pt.wikipedia.org/wiki/Simula%C3%A7%C3%A3o>

[2] Slides de aula: <http://www.comp.ita.br/~pauloac/>