Benchmark de cache hit e miss

Alan Thiago Santos de Oliveira

[1]Instituto de Computação – Universidade Federal do Amazonas (UFAM)
Av. General Rodrigo Octávio, 6200 – Coroado I – 69080-900 – Manaus – AM – Brasil

alan.thiago@icomp.ufam.edu.br

1. Introdução

A memória cache é uma memória de alta velocidade que trabalha em conjunto com o processador. Quando a CPU precisa acessar um dado na memória, ela solicita esse dado da cache. Caso esse dado não esteja na cache, ela carrega um bloco de palavras da memória RAM. Se o próximo dado acessado pela CPU estiver nesse bloco de palavras ocorre um *hit*, senão, ocorre um miss. Por isso, este relatório tem como objetivo analisar o crescimento da taxa de *cache-miss*, usando a ferramenta perf, quando aumentamos o intervalo de acesso entre indice do vetor

2. Materiais e métodos

2.1. Metodologia

Para realizar o experimento foi utilizado um código, em linguagem C, como *benchmark*. Esse programa percorre um vetor de tamanho $N * M$, de forma intercalada, onde M é o tamanho do intervalo de intercalação passado ao programa.

O experimento foi realizado utilizando a ferramenta perf para obter a taxa de *cache-miss* conforme o intervalo de intercalação do vetor aumenta. Com isso, foram utilizados 100 valores de intercalação distintos no intervalo [10, 1000], aumentando de 10 em 10. Para cada valor desse intervalo, o *benchmark* foi executado 10 vezes e a media aritmética do *cache-miss* foi computada. Após a obtenção das médias, foi criado um gráfico para para visualizar os o crescimento das médias das taxas de *cache-miss* conforme o intervalo aumenta.

2.2. Recursos de software

O *benchmark* foi compilado com GCC, versão 9.3.0, com a flag -o0 para evitar que o compilador fizesse otimizações no arquivo binário. Após isso, foi criado um *script* na linguagem Python, versão 3.8.5, para automatizar o processo de obtenção das médias de *cache-miss* e colocá-las num arquivo de texto. Em seguida, esses dados foram colocados no Google Planilhas para gerar o gráfico de crescimento das médias da taxa de *cache-miss*.

2.3. Plataforma de Hardware

O experimento foi executado em uma máquina com uma CPU Intel Core i5-5200U (2.7GHz), 8 GB de RAM, rodando Linux Mint 20. O processador tem dois núcleos arquitetura da CPU é x86_64. Cada núcleo tem tamanhos de chache L1, L2 e L3 de 64 KiB, 512 KiB e 512 KiB respectivamente.

3. Resultados

3.1. Intervalo de acessos

Como pode ser observado na figura 1, a taxa de *cache-miss* aumenta conforme cresce o intervalo entre os índices. Isso ocorre porque quando o processador carrega um elemento de um vetor para um de seus registradores, ele solicita esse dado da memória cache. Caso ocorra um *miss*, a cache carregará um bloco de dados consecutivos, com 1 ou mais palavras, da memória RAM. Esse bloco de dados é utilizado para ganhar desempenho com a localidade espacial. Pois, se o próximo dado solicitado pelo processador estiver dentro desse bloco, ocorrerá um *hit* e não será necessário carregar mais um bloco de dados da memória RAM. Portanto, se esse intervalo for muito grande, menor será a probabilidade do próximo elemento acessado estar no mesmo bloco que o anterior. Logo, maior será a taxa de *cache-miss*.

3.2. Ponto ótimo

Como pode ser observado no gráfico abaixo, o ponto ótimo é quando o intervalo de acesso é 10 pois ele tem a menor taxa de *cache-miss* (34,44%). Ele é um ponto ótimo pois tem o menor intervalo de acesso entre os índices do vetor. Logo, a sua taxa de *hits* é maior.

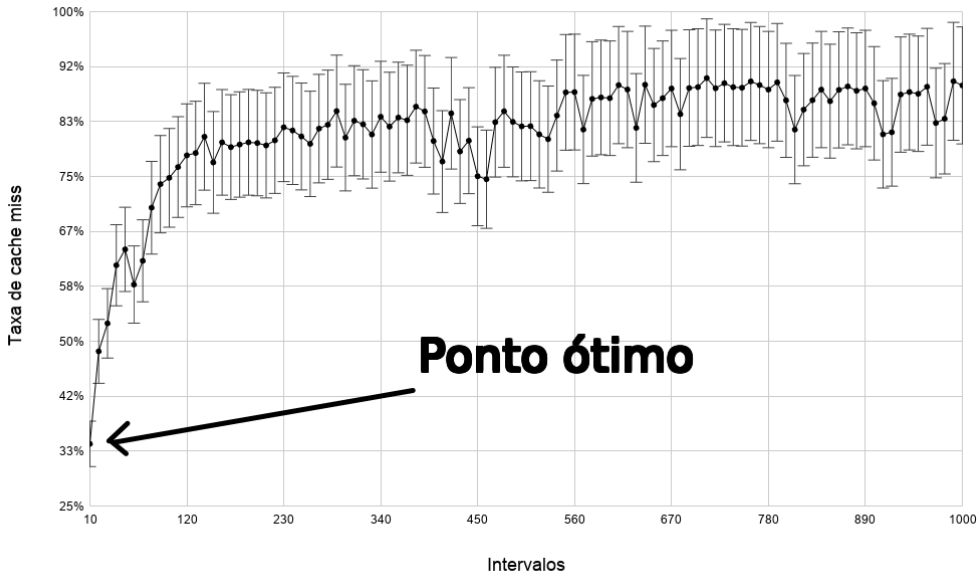

Figura 1. Percentual de *cache-miss* por intervalo de acesso do vetor

3.3. AMAT

Para um computador hipotético que possui somente um cache L1 e um memória RAM com tempos de acesso de 1 ciclo e 100 ciclos respectivamente, o AMAT calculado com a taxa do *miss-cache* do ponto ótimo é: $AMAT = 1 + 0.3444 \times 100 = 35.44$

4. Conclusão

Como pôde foi observado, quanto menor o intervalo de acessos entre índice de um vetor, maior será a taxa de *cashe-miss*. Consequentemente, menos blocos são carregados

da RAM fazendo com que o programa seja executado mais rapidamente. Logo, para obtermos 100% de ganho da localidade espacial, o programa deve percorrer o vetor em intervalos de 1.