Avaliando a memória cache

Organização de Computadores - CC53B

Elquias Monteiro da Costa Junior, 2386682

Resumo—A memória de cache é um intermediário entre a transferência de dados entre o processador e a memória principal, desenvolvida para melhorar o desempenho do acesso aos dados e instruções pelo processador, fazendo os acessos serem mais rápidos. No entanto, as características de uso da memória podem interferir na funcionalidade do cache, causar longos atrasos no sistema e afetar o desempenho do software em execução. Os experimentos propostos nesta tarefa mostram duas versões do mesmo programa para mostrar o impacto no desempenho da memória ao executar o software.

Index Terms—Organização de Computadores, Arquitetura de Computadores, Memória Cache, Vetores.

1 Introdução

O uso da memória cache nos permite acessar dados de forma rápida através das características de sua implementação, que consiste em guardar dados que possam ser utilizados num próximo momento, localidade temporal, e o princípio que faz a cache armazenar dados vizinhos para economizar visitas na memória principal, localidade espacial. Pelo fato da memória cache estar muito próxima do processador, ocorre a diminuição do tempo de acesso nessa memória, assim, isso acaba diminuindo muito o tempo que levaria para o processador acessar um dado através desse mediador. Atualmente, quando nos referenciamos à memória o termo cache é utilizado para referenciar qualquer aplicação que se beneficie dos princípios de localidade.

O objetivo deste trabalho é demonstrar através de experimentos, o quão eficiente é o programa ao manipular corretamente as características da memória cache.

2 Metodologia

No nosso problema, buscamos observar o funcionamento da memória cache, a partir disso, foram fornecidas duas funções que simulam o convertimento de uma imagem colorida para uma em escala de cinza, a primeira percorre o vetor de dados, como se fosse o pixel a ser convertido, de forma linear, e a segunda percorre de forma a pular blocos no seu acesso. Assim, essa mudança ocorre, na primeira, pelo acesso ser de cada elemento sequencialmente num índice de linhas e o segundo ocorre pelo acesso de cada elemento de um índice de colunas.

Desta forma, os princípios de memória cache conseguem ser utilizados na primeira função, ocorrem muito mais hits do que misses, e na segunda função é notável a busca em forçar as falhas na memória cache para que o sistema tenha que buscar os dados que o sistema pede na memória principal, o que leva ao acesso mais lento dos dados.

Para o desenvolvimento deste trabalho foi necessário a criação de um programa escrito na linguagem C, que nele colocamos as funções para a conversão de imagem junto a uma função que calcula quanto tempo um trecho de código demora a ser executado:

```
double ti,tf,tempo1; //ti = tempo1 inicial // tf = tempo1 final
ti = tf = tempo1 = 0;
struct timeval tempo_inicio, tempo_fim;
gettimeofday(&tempo_inicio,NULL);

int i,j;
for(i=0; i<TAM; i++){
    for(j=0; j<TAM; j++){
        color[i* TAM + j].r = (
        color[i* TAM + j].r +
        color[i* TAM + j].g +
        color[i* TAM + j].b
        )/3;
}
}

gettimeofday(&tempo_fim,NULL);
tf = (double)tempo_fim.tv_usec + ((double)tempo_fim.tv_sec * (1000000.0));
ti = (double)tempo_inicio.tv_usec + ((double)tempo_inicio.tv_sec * (1000000.0));
tempo1 = (tf - ti)/1000;
printf("Tempo1 gasto em milissegundos %.3f\n",tempo1);
```

Figura 1. Função 1 e função que calcula seu tempo de execução

E o mesmo foi feito para a segunda função logo abaixo do código da primeira:

```
double tempo2; //ti = tempo inicial // tf = tempo final
ti = tf = tempo2 = 0;
struct timeval tempo_inicio2, tempo_fim2;
gettimeofday(&tempo_inicio2,NULL);

i=j=0;
for(i=0; i<TAM; i++){
    for(j=0; j<TAM; j++){
        color[j* TAM + i].r = (
        color[j* TAM + i].r +
        color[j* TAM + i].g +
        color[j* TAM + i].b
        )/3;
}
}

gettimeofday(&tempo_fim2,NULL);
tf = (double)tempo_fim2.tv_usec + ((double)tempo_fim2.tv_sec * (1000000.0));
ti = (double)tempo_inicio2.tv_usec + ((double)tempo_inicio2.tv_sec * (1000000.0));
tempo2 = (tf - ti)/1000;
printf("Tempo2 gasto em milissegundos %.3f\n",tempo2);
```

Figura 2. Função 2 e função que calcula seu tempo de execução

2.1 Escrita no arquivo de números

```c
void criaNum(char *nomeArquivo){
    FILE *arq;
    arq = fopen(nomeArquivo, "wb");
    int i;
    for(i=1000; i<=250000; i+=10){
            fwrite(&i, sizeof(int), 1, arq);
    }
    fclose(arq);
}
```

Figura 3. Função criaNum

Essa função abre/cria um arquivo binário e escreve o intervalo dado na função for, nesse caso ele inicia em 1000 e acrescenta de 10 em 10 até chegar em 250000

2.2 Leitura de arquivo binário

```c
int leNum(char *nomeArquivo){
    FILE *arq;
    int *elemento = NULL, quantidade = 0;
    arq = fopen(nomeArquivo, "rb");
    elemento = (int *)malloc(quantidade * sizeof(int));

    if (arq == NULL) {
    //Sai do programa caso o arquivo não tenha sido aberto
    printf("Erro ao tentar abrir o arquivo.\n");
    exit(1);
    }
    //Efetua a contagem de elementos lendo o arquivo
     while (fread(elemento, 4, 1, arq) == 1){
        //printf("%d\n", *elemento);
        if (elemento != NULL) {
        quantidade++;
      }
    }

    //Fecha o arquivo
    fclose(arq);
    free(elemento);
    //Exibe a quantidade de linhas
    printf("Quantidade de elementos: %d\n",quantidade);
    return quantidade;
}
```

Figura 4. Função leNum

A função chamada leNum realiza a leitura de cada elemento escrito no arquivo binário e adiciona no contador “quantidade” quando a operação é um sucesso, e retorna ao final da função.

2.3 Escrita dos resultados no arquivo

```c
//tempo em um arquivo
FILE *arq;
arq = fopen("tempo.txt", "a+b");
fprintf(arq, "%d %.3f %.3f\n", TAM, tempo1, tempo2);
fclose(arq);
```

Figura 5. Trecho que escreve em arquivo

Esse trecho da função está logo após as funções que simulam a conversão de imagem, ela escreve em um arquivo de texto o tempo que as funções 1 e 2 levaram para serem executadas, juntamente com o tamanho usado na alocação do vetor.

3 Automação dos testes e resultados

Para iniciar a automatização, se viu necessário uma forma de definir previamente a variação do tamanho para dar de entrada em cada chamada da main do arquivo C que simula a conversão de imagem. Para isso foi criado a função descrita no tópico 2.1. Após isso, notou-se a necessidade de contar a quantidade de número escritos na no arquivo binário criados pela função criaNum, essa função foi descrita no tópico 2.2. Essa função que retorna o tamanho também nos auxilia quando plotamos o gráfico, o tamanho é um argumento utilizado para fazer a leitura dos dados contidos num arquivo.

3.1 Função main da automação

```c
int main(){
    criaNum("Verynums.bin");
    int tamanho = leNum("Verynums.bin"), elemento = 0;

    system("rm tempo.txt");

    FILE *arquivo = fopen("tempo.txt", "a+b");
    fprintf(arquivo, "%d\n",tamanho);
    fclose(arquivo);

    FILE *arq = fopen("Verynums.bin", "rb");
    char var[100];

        while(fread(&elemento, 4, 1, arq) == 1){
            //printf("%d\n", elemento);
            sprintf(var, "gcc orgTrab1.c && ./a.out %d", elemento);
            system(var); //system recebe valor de var
        }

    system("octave");
    fclose(arq);
    return 0;
}
```

Figura 3. Main da automação

Para realizar o processo de automação, primeiramente, foi necessário chamar a função criaNum para criar um arquivo binário de inteiros que serão os tamanhos. Logo após, é declarada a variável inteira “tamanho” que vai receber o retorno da função leNum, que é a quantidade de elementos contidos no arquivo chamado “Verynums.bin”. Ademais, o comando “system(“rm tempo.txt”)” deleta outros arquivos “tempo.txt” existentes para facilitar os testes. Em seguida, ocorre a escrita no arquivo “tempo.txt” com o tamanho vindo de retorno da função leNum para poder ser usado mais a frente quando formos plotar o gráfico. Então, é criado uma variável char de 100 posições, além disso, o programa cai na estrutura de Controle while, a condição que mantém o repetição é o sucesso da leitura de um inteiro da arquivo de tamanhos no comando fread, enquanto a condição for verdadeira ocorrerá um sprintf, esse comando funciona como se fosse feita a disposição do printf, porém, alocando a string no vetor char “var” o retorno da função. Na variável var foi alocado o comando do gcc para compilar o arquivo “orgTrab1.c” junto com o executável resultante do comando gcc junto ao “elemento”, que é o tamanho que varia em cada loop de while.

```c
int main(int argc, char *argv[]){

    int TAM = atoi(argv[1]);
    //int TAM = 1000;

    color = (pix *) malloc(TAM * TAM *sizeof(pix));
```

Figura 6. Main das funções que simulam a conversão

O “elemento” chega como parâmetro da main, dessa forma acontece a mudança do “TAM” da main, e após isso, ela é usada para alocar memória com o malloc, todas as variáveis alocadas dinamicamente são liberadas pelo

programa. Após várias chamadas do while é formado o arquivo .txt que contém na primeira coluna os tamanhos, na segunda o tempo da função 1 e na terceira o tempo da função 2.

3.2 Função para plotar gráfico no octave

```
function plotGrafico()
  arq = fopen('tempo.txt', 'r');
 if(arq == -1)
     printf('Erro: nao foi possivel abrir o arquivo');
     return;
 endif
tamanho = fscanf(arq, '%d', [1 1]);
indices = fscanf(arq, '%f %f %f', [3 tamanho]);
indices = indices';

fclose(arq);

x = indices(:,1);
y = indices(:,2);
z = indices(:,3);

plot(x,y,'r',x,z,'b');
xlabel("Tamanhos");
ylabel("Tempo em milissegundos");
title("Graficos desempenho de cache");

endfunction
```

Figura 7. Script da função no octave

Foi-se criado utilizando a ferramenta octave uma função que realiza a leitura do arquivo de nome “tempo.txt” e extrai o tamanho da matriz e as 3 colunas necessárias para montar o gráfico. Para utilizar essa função é necessário ter o Octave instalado, ao final da execução o código abre o Octave no seu terminal, então, o usuário precisa digitar “plotGrafico” para executar o script e ter como saída um gráfico das duas funções na ferramenta octave.

4 Experimentos

Para a realização de experimentos foi se utilizado uma máquina com processador Intel® Core™ i5-10300H CPU @ 2.50GHz × 8, com 256KB de cache L1, 1MB de cache L2, 8MB de cache L3, com 8GB de memória ram e com o sistema operacional Ubuntu 20.04.4 LTS de 64 bits.

Os dados da operação da função foram salvos no arquivo “tempo.txt” no qual é possível ver todos os tempos para cada tamanho testado. Foram realizados 2 testes para ver o limite da função, o primeiro teste foi realizado com o tamanho variando de 1.000 a 23.500 (limite do tamanho permitido pela minha máquina) com um incremento de 100 números por cada chamada da função, a cor vermelha representa a função 1 e cor azul representa a função 2, o gráfico resultante foi o seguinte.

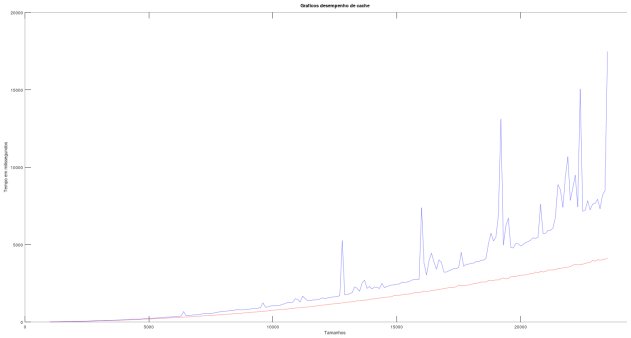

Figura 8. Gráfico de desempenho 1

E no teste 2 foi realizado com o tamanho indo de 1.000 a 24.030 com incremento de 10 números por cada chamada da função, e o gráfico resultante foi o seguinte.

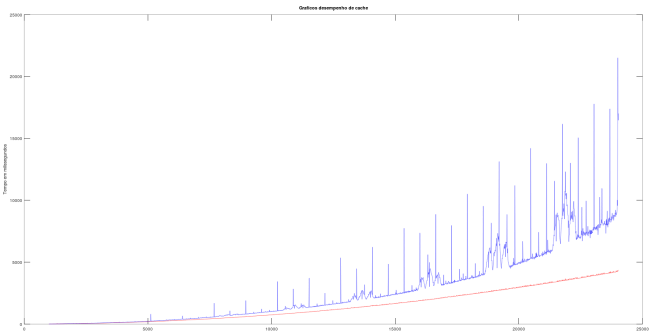

Figura 9. Gráfico de desempenho 2

5 Conclusão

Analisando os resultados obtidos, podemos notar que o desempenho do uso correto da cache, a função de cor vermelha, é quase linear e com poucas variações. No entanto, a função 2 que força a falha da memória cache tem muitas instabilidades, pois é como se a função apenas se comunicasse com a memória principal para realizar as operações que foram solicitadas. Assim, quando manipulamos uma grande massa de dados em vetores, utilizar a cache corretamente nos garante grande economia de tempo nos nossos acessos à memória.

Ademais, suponho que a causa dos testes não conseguirem ser realizados com tamanhos maiores do que 25.000 seja devido ao tamanho da minha memória ram, que talvez a função malloc tenha solicitado um espaço de memória que o sistema operacional não tenha conseguido entregar.

Bibliografia

[1] Read Data from text and csv Files using GNU Octave and Matlab.
https://www.youtube.com/watch?v=cLo2UOBU5yY

[2] Curso de Octave - Múltiplos Gráficos #19
https://www.youtube.com/watch?v=DAnVdVyGFVs

[3] USANDO COMANDOS DO SISTEMA EM C
https://www.vivaolinux.com.br/dica/Usando-comandos-do-sistema-em-C#:~:text=Pois%20bem%2C%20para%20colocar%20comando,Ok%2C%20at%C3%A9%20a%C3%AD%20tudo%20bem!

[4] Laço para leitura de arquivo de texto lê a última linha duas vezes
https://pt.stackoverflow.com/questions/269714/la%C3%A7o-para-leitura-de-arquivo-de-texto-l%C3%AA-a-%C3%BAltima-linha-duas-vezes

[5] Não entendi muito bem a função `sprintf`
https://cursos.alura.com.br/forum/topico-nao-entendi-muito-bem-a-funcao-sprintf-35902