

# Atividade Prática 4

Iago Zagnoli Albergaria - 2022069476

Universidade Federal de Minas Gerais (UFMG)  
Belo Horizonte, MG - Brasil

[iagozag@gmail.com](mailto:iagozag@gmail.com)

## Introdução

O programa a ser analisado possui um conjunto de funções para operações em matrizes, sendo elas criar uma matriz(nula ou com valores aleatórios), transpô-la ou gerar uma nova como resultado da soma/multiplicação de outras duas.

**mat.h:**

```
C/C++  
#ifndef MATH  
#define MATH  
  
#ifndef MAXTAM  
#define MAXTAM 500  
#endif  
  
typedef struct mat{  
    double m[MAXTAM][MAXTAM];  
    int tamx, tamy;  
    int id;  
} mat_tipo;  
  
void criaMatriz(mat_tipo * mat, int tx, int ty, int id);  
void inicializaMatrizNula(mat_tipo * mat);  
void inicializaMatrizAleatoria(mat_tipo * mat);  
double acessaMatriz(mat_tipo * mat);  
void imprimeMatriz(mat_tipo * mat);  
void salvaMatriz(mat_tipo * mat, FILE *);  
void escreveElemento(mat_tipo * mat, int x, int y, double v);  
double leElemento (mat_tipo * mat, int x, int y);  
void copiaMatriz(mat_tipo * src, mat_tipo * dst, int dst_id);  
void somaMatrizes(mat_tipo * a, mat_tipo * b, mat_tipo * c);  
void multiplicaMatrizes(mat_tipo * a, mat_tipo * b, mat_tipo * c);  
void transpoeMatriz(mat_tipo *a);  
void destroiMatriz(mat_tipo *a);  
  
#endif
```

## mat.c:

```
C/C++  
  
#include <stdlib.h>  
#include <stdio.h>  
#include <math.h>  
#include "mat.h"  
#include "msgassert.h"  
  
// limite superior da inicializacao aleatoria  
#define INITRANDOMRANGE 10  
// Macro que realiza swap sem variavel auxiliar  
#define ELEMSWAP(x,y) (x+=y,y=x-y,x-=y)  
  
void criaMatriz(mat_tipo * mat, int tx, int ty, int id)  
// Descricao: cria matriz com dimensoes tx X ty  
// Entrada: mat, tx, ty, id  
// Saída: mat  
{  
    // verifica se os valores de tx e ty são validos  
    erroAssert(tx>0,"Dimensao nula");  
    erroAssert(ty>0,"Dimensao nula");  
    erroAssert(tx<=MAXTAM,"Dimensao maior que permitido");  
    erroAssert(ty<=MAXTAM,"Dimensao maior que permitido");  
  
    // inicializa as dimensoes da matriz  
    mat->tamx = tx;  
    mat->tamy = ty;  
    // inicializa o identificador da matriz, para rastreamento  
    mat->id = id;  
}  
  
void inicializaMatrizNula(mat_tipo * mat)  
// Descricao: inicializa mat com valores nulos  
// Entrada: mat  
// Saída: mat  
{  
    int i, j;  
    // inicializa todos os elementos da matriz com 0, por segurança  
    for (i=0; i<MAXTAM; i++){  
        for(j=0; j<MAXTAM; j++){  
            mat->m[i][j] = 0;  
        }  
    }  
}  
  
void inicializaMatrizAleatoria(mat_tipo * mat)  
// Descricao: inicializa mat com valores aleatorios  
// Entrada: mat  
// Saída: mat
```

```

{
    int i, j;
    // inicializa a matriz com valores nulos, por seguranca
    inicializaMatrizNula(mat);
    // inicializa a parte alocada da matriz com valores aleatorios
    for (i=0; i<mat->tamx; i++){
        for(j=0; j<mat->tamy; j++){
            mat->m[i][j] = drand48()*INITRANDOMRANGE;
        }
    }
}

void imprimeMatriz(mat_tipo * mat)
// Descricao: imprime a matriz com a identificacao de linhas e colunas
// Entrada: mat
// Saída: impressao na saida padrao (stdout)
{
    int i,j;

    // segurança, mas erro não deve acontecer jamais
    erroAssert(mat->tamx<=MAXTAM, "Dimensao maior que permitido");
    erroAssert(mat->tamy<=MAXTAM, "Dimensao maior que permitido");

    // imprime os identificadores de coluna
    printf("%9s", " ");
    for(j=0; j<mat->tamy; j++){
        printf("%8d ",j);
    }
    printf("\n");

    // imprime as linhas
    for (i=0; i<mat->tamx; i++){
        printf("%8d ",i);
        for(j=0; j<mat->tamy; j++){
            printf("%8.2f ",mat->m[i][j]);
        }
        printf("\n");
    }
}

void salvaMatriz(mat_tipo * mat, FILE * out)
// Descricao: salva a matriz em arquivo
// Entrada: mat
// Saída: out
{
    int i,j;

    // segurança, mas erro não deve acontecer jamais
    erroAssert(mat->tamx<=MAXTAM, "Dimensao maior que permitido");

```

```

erroAssert(mat->tamy<=MAXTAM, "Dimensao maior que permitido");

fprintf(out, "%d %d\n", mat->tamx, mat->tamy);
// imprime as linhas
for (i=0; i<mat->tamx; i++){
    for(j=0; j<mat->tamy; j++){
        fprintf(out, "% .6f ", mat->m[i][j]);
    }
    fprintf(out, "\n");
}
}

void somaMatrizes(mat_tipo *a, mat_tipo *b, mat_tipo *c)
// Descricao: soma as matrizes a e b e armazena o resultado em c
// Entrada: a, b
// Saída: c
{
    int i,j;
    // verifica se as dimensoes das matrizes a e b sao as mesmas
    erroAssert(a->tamx==b->tamx, "Dimensoes incompatíveis");
    erroAssert(a->tamy==b->tamy, "Dimensoes incompatíveis");

    // inicializa a matriz c garantindo a compatibilidade das dimensoes
    criaMatriz(c,a->tamx, a->tamy, c->id);
    inicializaMatrizNula(c);

    // faz a soma elemento a elemento
    for (i=0; i<a->tamx; i++){
        for(j=0; j<a->tamy; j++){
            c->m[i][j] = a->m[i][j]+b->m[i][j];
        }
    }
}

void multiplicaMatrizes(mat_tipo *a, mat_tipo *b, mat_tipo *c)
// Descricao: multiplica as matrizes a e b e armazena o resultado em c
// Entrada: a,b
// Saída: c
{
    int i,j,k;
    // verifica a compatibilidade das dimensoes
    erroAssert(a->tamy==b->tamx, "Dimensoes incompatíveis");

    // cria e inicializa a matriz c
    criaMatriz(c,a->tamx, b->tamy,c->id);
    inicializaMatrizNula(c);

    // realiza a multiplicacao de matrizes
}

```

```

for (i=0; i<c->tamx;i++){
    for (j=0; j<c->tamy;j++){
        for (k=0; k<a->tamy;k++){
            c->m[i][j] += a->m[i][k]*b->m[k][j];
        }
    }
}

void transpoeMatriz(mat_tipo *a)
// Descricao: transpoe a matriz a
// Entrada: a
// Saída: a
{
    int i, j, dim;

    // determina a maior dimensao entre tamx e tamy
    dim = (a->tamx>a->tamy?a->tamx:a->tamy);

    // faz a transposicao como se a matriz fosse quadrada
    for (i=0; i<dim; i++){
        for(j=i+1; j<dim; j++){
            ELEMSWAP((a->m[i][j]),(a->m[j][i]));
        }
    }
    // inverte as dimensoes da matriz transposta
    ELEMSWAP(a->tamx,a->tamy);
}

void destroiMatriz(mat_tipo *a)
// Descricao: destroi a matriz a, que se torna inacessivel
// Entrada: a
// Saída: a
{
    // apenas um aviso se a matriz for destruida mais de uma vez
    avisoAssert(((a->tamx>0)&&(a->tamy>0)), "Matriz já foi destruída");

    // torna as dimensoes invalidas
    a->id = a->tamx = a->tamy = -1;
}

```

## matop.c:

```
C/C++  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <math.h>  
#include <getopt.h>  
#include <string.h>  
#include "mat.h"  
#include "msgassert.h"  
  
// definicoes de operacoes a serem testadas  
#define OPSOMAR 1  
#define OPMULTIPLICAR 2  
#define OPTRANSPOR 3  
#define OPCRIAR 4  
  
// variaveis globais para opcoes  
static int opescolhida;  
char outname[100];  
int optx, opty;  
  
void uso()  
// Descricao: imprime as opcoes de uso  
// Entrada: nao tem  
// Saída: impressao das opcoes de linha de comando  
{  
    fprintf(stderr, "matop\n");  
    fprintf(stderr, "\t-s \t(somar matrizes) \n");  
    fprintf(stderr, "\t-m \t(multiplicar matrizes) \n");  
    fprintf(stderr, "\t-t \t(transpor matriz)\n");  
    fprintf(stderr, "\t-c <arq> \t(cria matriz e salva em arq)\n");  
    fprintf(stderr, "\t-x <int>\t(primeira dimensao)\n");  
    fprintf(stderr, "\t-y <int>\t(segunda dimensao)\n");  
}  
  
void parse_args(int argc, char ** argv)  
// Descricao: le as opcoes da linha de comando e inicializa variaveis  
// Entrada: argc e argv  
// Saída: opescolhida, optx, opty  
{  
    // variaveis externas do getopt  
    extern char * optarg;  
    extern int optind;  
  
    // variavel auxiliar  
    int c;  
  
    // inicializacao variaveis globais para opcoes  
    opescolhida = -1;
```

```

optx = -1;
opty = -1;
outnome[0] = 0;

// getopt - letra indica a opcao, : junto a letra indica parametro
// no caso de escolher mais de uma operacao, vale a ultima
while ((c = getopt(argc, argv, "smtc:p:x:y:lh")) != EOF){
    switch(c) {
        case 'm':
            avisoAssert(opescolhida== -1, "Mais de uma operacao escolhida");
            opescolhida = OPMULTIPLICAR;
            break;
        case 's':
            avisoAssert(opescolhida== -1, "Mais de uma operacao escolhida");
            opescolhida = OPSOMAR;
            break;
        case 't':
            avisoAssert(opescolhida== -1, "Mais de uma operacao escolhida");
            opescolhida = OPTRANSPOR;
            break;
        case 'c':
            avisoAssert(opescolhida== -1, "Mais de uma operacao escolhida");
            opescolhida = OPCRIAR;
            strcpy(outnome, optarg);
            break;
        case 'x':
            optx = atoi(optarg);
            break;
        case 'y':
            opty = atoi(optarg);
            break;
        case 'h':
        default:
            uso();
            exit(1);

    }
}

// verificacao da consistencia das opcoes
erroAssert(opescolhida>0, "matop - necessario escolher operacao");
erroAssert(optx>0, "matop - dimensao X da matriz tem que ser
positiva");
erroAssert(opty>0, "matop - dimensao Y da matriz tem que ser
positiva");
if (opescolhida==OPCRIAR){
    erroAssert(strlen(outnome)>0, "matop - nome de arquivo de saida tem
que ser definido");
}

```

```

}

int main(int argc, char ** argv)
// Descricao: programa principal para execucao de operacoes de matrizes
// Entrada: argc e argv
// Saída: depende da operacao escolhida
{
    // ate 3 matrizes sao utilizadas, dependendo da operacao
    mat_tipo a, b, c;
    FILE * outfile;

    // avaliar linha de comando
    parse_args(argc,argv);

    // execucao dependente da operacao escolhida
    switch (opescolhida){
        case OPSOMAR:
            // cria matrizes a e b aleatorias, que sao somadas para a matriz c
            // matriz c é impressa e todas as matrizes sao destruidas
            criaMatriz(&a,optx,opty,0);
            inicializaMatrizAleatoria(&a);
            criaMatriz(&b,optx,opty,1);
            inicializaMatrizAleatoria(&b);
            criaMatriz(&c,optx,opty,2);
            inicializaMatrizNula(&c);
            somaMatrizes(&a,&b,&c);
            imprimeMatriz(&c);
            destroiMatriz(&a);
            destroiMatriz(&b);
            destroiMatriz(&c);
            break;
        case OPMULTIPLICAR:
            // cria matrizes a e b aleatorias, que sao multiplicadas para matriz c
            // matriz c é impressa e todas as matrizes sao destruidas
            criaMatriz(&a,optx,opty,0);
            inicializaMatrizAleatoria(&a);
            criaMatriz(&b,opty,optx,1);
            inicializaMatrizAleatoria(&b);
            criaMatriz(&c,optx,optx,2);
            inicializaMatrizNula(&c);
            multiplicaMatrizes(&a,&b,&c);
            imprimeMatriz(&c);
            destroiMatriz(&a);
            destroiMatriz(&b);
            destroiMatriz(&c);
            break;
        case OPTRANSPOR:
            // cria matriz a aleatoria, que e transposta, impressa e destruida
    }
}

```

```

        criaMatriz(&a,optx,opty,0);
        inicializaMatrizAleatoria(&a);
        transpoeMatriz(&a);
        imprimeMatriz(&a);
        destroiMatriz(&a);
        break;
    case OPCRIAR:
        // cria matriz a aleatoria, que e salva
        outfile = fopen(outnome, "wt");
        erroAssert(outfile != NULL, "Erro na criacao do arquivo de saida");
        criaMatriz(&a,optx,opty,0);
        inicializaMatrizAleatoria(&a);
        salvaMatriz(&a,outfile);
        destroiMatriz(&a);
        fclose(outfile);
        break;
    default:
        // nao deve ser executado, pois ha um erroAssert em parse_args
        uso();
        exit(1);
    }

    // conclui registro de acesso
    return 0;
}

```

## 1. Os acessos de memória esperados do programa estão nas seguintes funções:

- **criaMatriz**: Quando a função é invocada, as variáveis de dimensionamento e identificação da matriz são assinaladas de acordo com os parâmetros da função.
- **inicializaMatrizNula**: A função percorre todos os elementos da matriz e inicializa cada posição com o valor '0'. Para cada elemento, é necessário um acesso na memória.
- **inicializaMatrizAleatoria**: De forma semelhante à função anterior, a função também acessa na memória cada elemento da matriz, mas em vez de '0', inicializa os elementos com valores pseudo-aleatórios, que também precisam de acessos na memória para serem gerados.
- **imprimeMatriz, salvaMatriz**: Ambas as funções acessam a memória para recuperar cada elemento da matriz e printar. A função salvaMatriz ainda tem de acessar um arquivo na memória para escrever os valores da matriz.
- **somaMatrizes, multiplicaMatrizes, transpoeMatriz**: As três funções precisam de acessar os elementos das matrizes que necessitam para realizar suas respectivas operações. As funções de soma e multiplicação acessam a

memória das duas matrizes dadas como parâmetro e escrevem os resultados de cada posição numa terceira matriz. A função acessa a memória para trocar os elementos da matriz para que obtenha a transposta da matriz inicial.

- **destroiMatriz**: Acessa as variáveis de dimensão e identificador da matriz na memória e altera seus valores.

**Localidade de referência**: Todas as posições estão localizadas de forma sequencial e contínua na memória por se tratar de vetores, por isso a localidade espacial é aproveitada no código. Quando um elemento é acessado em um determinado momento, a probabilidade das próximas posições da matriz também serem acessadas na memória aumenta. Diferentemente, a localidade temporal não é tão bem utilizada no código, haja vista que a cada operação de matriz, é gerada uma nova matriz pseudo-aleatória para que se obtenha um resultado. Caso existisse um conjunto de matrizes que fossem utilizadas recorrentemente em determinado algoritmo, a localidade temporal seria melhor utilizada, já que a probabilidade de uma determinada posição na memória ser acessada novamente no futuro aumentaria.

**Estruturas de dados críticas**: A estrutura de dados utilizada no código em discussão é uma struct “mat”, com uma definição de matriz de tamanho máximo 50 para ambas linhas e colunas. Além disso, também tem dois inteiros, tamx e tamy, respectivamente o tamanho em linhas e colunas da matriz e um identificador. Cada elemento individualmente também é crítico, pois é modificado o tempo todo.

**Segmentos de código críticos**: Os segmentos mais críticos do código são aqueles que realizam as operações com as matrizes: soma, multiplicação e transposição. Isso se dá, pois a quantidade de acessos à memória são significativos, ainda mais se dadas matrizes muito grandes como parâmetro. A operação de multiplicação, por exemplo, tem custo de  $O(n^3)$ , ou seja, acessa todos os elementos da matriz 3 vezes a cada vez que é chamada.

**2.** Para o plano de caracterização de localidade de referência do código, serão utilizadas as ferramentas cachegrind e callgrind, ambas flags possíveis na execução do programa valgrind.

O cachegrind traça um perfil com alta precisão do programa medindo a quantidade exata de instruções executadas e apresenta os dados em nível de arquivo, função e linha. Em geral, tempo de execução é uma métrica melhor do que um contador de instruções, porque é o que o usuário percebe no geral, mas pode acabar variando mesmo que o programa tenha a mesma entrada. Ao contrário, contadores de instruções são mais reproduzíveis, ou seja, pequenas mudanças em um programa podem ser medidas com alta precisão.

Já o callgrind é uma ferramenta que grava o histórico de chamadas de funções na execução de um programa como um call-graph. Os dados coletados

consistem no número de instruções executadas, suas relações com as linhas do código, quem chamou e quem foi chamado entre as funções e a quantidade dessas chamadas.

**3.** Serão executadas chamadas da função multiplicaMatriz com matrizes com dimensão de 100 linhas/colunas para que o comportamento do programa seja verificado para ambos, cachegrind e callgrind.

#### 4. Cachegrind:

```
$ valgrind --tool=cachegrind bin/matop -m -x 100 -y 100
==41460== CacheGrind, a cache and branch-prediction profiler
==41460== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==41460== Using Valgrind-3.21.0 and LibVEX; rerun with -h for copyright info
==41460== Command: bin/matop -m -x 100 -y 100
==41460==
```

| 0       | 1       | 2       | 3       | 4       | 5       | 6       | 7       | 8       | 9       | 10      | 11      | 12      |         |
|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 13      | 14      | 15      | 16      | 17      | 18      | 19      | 20      | 21      | 22      | 23      | 24      | 25      | 26      |
| 27      | 28      | 29      | 30      | 31      | 32      | 33      | 34      | 35      | 36      | 37      | 38      | 39      | 40      |
| 41      | 42      | 43      | 44      | 45      | 46      | 47      | 48      | 49      | 50      | 51      | 52      | 53      | 54      |
| 56      | 57      | 58      | 59      | 60      | 61      | 62      | 63      | 64      | 65      | 66      | 67      | 68      | 69      |
| 70      | 71      | 72      | 73      | 74      | 75      | 76      | 77      | 78      | 79      | 80      | 81      | 82      | 83      |
| 84      | 85      | 86      | 87      | 88      | 89      | 90      | 91      | 92      | 93      | 94      | 95      | 96      | 97      |
| 98      | 99      |         |         |         |         |         |         |         |         |         |         |         |         |
| 0       | 2357.71 | 2451.76 | 2134.93 | 2260.74 | 2201.76 | 2534.09 | 2437.88 | 2482.72 | 2177.95 | 2287.96 | 2240.65 | 2765.53 | 2352.69 |
| 177.45  | 2394.90 | 2543.28 | 2090.36 | 2514.53 | 2597.81 | 2404.50 | 2243.11 | 2219.89 | 2574.92 | 2167.88 | 2127.55 | 2375.62 | 2364.14 |
| 3.34    | 2213.35 | 2122.63 | 2429.13 | 2490.08 | 2342.46 | 2500.33 | 2530.21 | 2441.39 | 2384.55 | 2752.00 | 2650.02 | 2291.87 | 2350.23 |
| 94      | 2455.21 | 2527.99 | 2112.30 | 2294.27 | 2159.13 | 2357.05 | 2386.82 | 2466.82 | 2291.34 | 2532.72 | 2385.20 | 2196.05 | 2581.32 |
| 2687.42 | 2666.45 | 2052.49 | 2373.29 | 2288.00 | 2111.87 | 2243.64 | 2377.55 | 2281.14 | 2187.01 | 2468.00 | 2829.27 | 2234.10 | 2479.81 |
| 2525.95 | 2320.04 | 2271.66 | 2356.12 | 2324.44 | 2230.48 | 2159.46 | 2636.47 | 2545.32 | 2250.34 | 2258.86 | 2479.82 | 2555.97 | 2287.38 |
| 24      | 17.32   | 2129.45 | 2693.14 | 2185.52 | 2306.61 | 2292.79 | 2377.77 | 2233.83 | 2360.08 | 2310.77 | 2538.26 | 2323.00 | 2001.50 |
| .00     | 2213.09 |         |         |         |         |         |         |         |         |         |         |         |         |
| 1       | 2676.87 | 2664.43 | 2557.38 | 2483.32 | 2277.17 | 2689.49 | 2644.73 | 2674.33 | 2365.69 | 2389.06 | 2694.18 | 2862.87 | 2662.20 |
| 2       | 480.96  | 2548.10 | 2692.73 | 2181.51 | 2627.95 | 2636.21 | 2440.75 | 2419.41 | 2400.62 | 2570.53 | 2381.76 | 2431.49 | 2677.21 |
|         |         |         |         |         |         |         |         |         |         |         |         |         | 2685.56 |
|         |         |         |         |         |         |         |         |         |         |         |         |         | 268     |

```
99 2733.75 2772.12 2421.56 2607.24 2330.56 2605.11 2595.99 2756.10 2461.76 2344.17 2553.12 2945.19 2672.31 2
263.88 2604.95 2703.55 2086.69 2548.01 2561.33 2611.29 2342.36 2486.49 2574.63 2306.60 2493.65 2789.36 2685.75 258
7.83 2589.03 2307.18 2521.27 2649.75 2520.88 2604.26 2642.04 2534.24 2699.31 2648.36 2759.27 2503.03 2543.17 2376.
57 2483.85 2693.74 2305.77 2569.72 2587.69 2758.67 2549.18 2595.36 2523.52 2613.32 2427.19 2421.53 2876.61 2539.40
2734.71 2736.05 2389.45 2765.96 2579.76 2300.36 2261.68 2677.39 2511.12 2414.81 2629.07 2959.19 2596.28 2697.91
2622.19 2573.06 2556.31 2625.35 2595.00 2378.04 2386.33 2925.50 2702.95 2548.29 2448.71 2576.48 2453.93 2371.03 25
17.81 2285.65 2767.88 2282.92 2586.02 2467.92 2569.74 2712.13 2761.72 2469.91 2470.01 2614.04 2226.64 2522.04 2685
.25 2264.80
==67896==
```

==67896== I refs: 79,263,288

==67896== I1 misses: 1,555

==67896== LLi misses: 1,526

==67896== I1 miss rate: 0.00%

==67896== LLi miss rate: 0.00%

==67896==

==67896== D refs: 37,132,375 (31,048,124 rd + 6,084,251 wr)

==67896== D1 misses: 268,845 ( 140,659 rd + 128,186 wr)

==67896== LLd misses: 95,419 ( 1,319 rd + 94,100 wr)

==67896== D1 miss rate: 0.7% ( 0.5% + 2.1% )

==67896== LLd miss rate: 0.3% ( 0.0% + 1.5% )

==67896==

==67896== LL refs: 270,400 ( 142,214 rd + 128,186 wr)

==67896== LL misses: 96,945 ( 2,845 rd + 94,100 wr)

==67896== LL miss rate: 0.1% ( 0.0% + 1.5% )

**flag –cache=yes=sim**

```
-- Metadata
Invocation: /sbin/cg_annotate cachegrind.out.67896
I1 cache: 32768 B, 64 B, 8-way associative
D1 cache: 32768 B, 64 B, 8-way associative
L1 cache: 6291456 B, 64 B, 12-way associative
Command: bin/matop -m -x 100 -y 100
Events recorded: Ir IImr IImr Dr DImr DImr Dw DImw DImw
Events shown: Ir IImr IImr Dr DImr DImr Dw DImw DImw
Event sort order: Ir IImr IImr Dr DImr DImr DImr Dw DImw DImw
Threshold: 0.1%
Annotation: on

-- Summary
Ir_ IImr_ IImr_ Dr_ DImr_ DImr_ Dw_ DImw_ DImw_
-----
79,263,288 (100.0%) 1,555 (100.0%) 1,526 (100.0%) 31,048,124 (100.0%) 140,659 (100.0%) 1,319 (100.0%) 6,084,251 (100.0%) 128,186 (100.0%) 94,180 (100.0%) PROGRAM TOTALS

-- File:function summary
Ir_ IImr_ IImr_ IImr_ Dr_ DImr_ DImr_ DImr_ Dw_
DImw_ DImw_ file:function DImw_ DImw_ DImw_ Dw_
-----
< 51,630,631 (65.1%, 65.1%) 26 (1.7%, 1.7%) 26 (1.7%, 1.7%) 24,319,540 (78.3%, 78.3%) 138,808 (98.7%, 98.7%) 1 (0.1%, 0.1%) 2,062,780 (33.9%, 33.9%) 127,657 (99.6%, 99.6%) 93,645 (99.5%, 99.5%) /home/iago/Documents/school/ufmg/3.Semestre/ED/codes/TP4/src/mat.c: 39,111,134 (49.3%) 7 (0.5%) 7 (0.5%) 19,070,717 (61.4%) 137,503 (97.8%) 0 1,010,107 (16.6%) 1 (0.8%) 0 multiplicacionMatrices 5,006,012 (16.1%) 0 0 0 1,002,012 (16.5%) 12,014,044 (15.2%) 0 0 inicializaMatrizNula 322,234 (0.4%) 93,643 (99.5%) 3 (0.2%) 161,412 (0.5%) 3 (0.0%) 1 (0.1%) 48,208 (0.7%) 2,652 (2.1%) 0 inicializaMatrizAleatoria
```

**flag -cache-yes=no**

```
$ cg_annotate cachegrind.out.41460
-- Metadata
Invocation: /sbin/cg_annotate cachegrind.out.41460
Command: bin/matop -m -x 100 -y 100
Events recorded: Ir
Events shown: Ir
Event sort order: Ir
Threshold: 0.1%
Annotation: on

-- Summary
Ir
79,263,288 (100.0%) PROGRAM TOTALS

-- File:function summary
Ir----- file:function
< 51,630,631 (05.1%, 65.1%) /home/iago/Documents/school/ufmg/3_Semestre/ED/codes/TP4/src/mat.c:
39,111,154 (49.3%) multiplicaMatrizes
12,014,044 (15.2%) inicializaMatrizNula
322,234 (0.4%) inicializaMatrizAleatoria
183,036 (0.2%) imprimeMatriz

< 10,939,944 (13.8%, 78.9%) /usr/src/debug/glibc/glibc/stdio-common/printf_fp.c:
6,781,979 (8.6%) __printf_fp_buffer_1.isra.0
3,607,965 (4.6%) hack_digit
550,000 (0.7%) __printf_fp_l_buffer

< 3,166,098 (4.0%, 82.9%) /usr/src/debug/glibc/glibc/libc/libc.so.6:__mpn_divrem
```

```
-- Annotated source file: /home/iago/Documents/school/ufmg/3_Semestre/ED/codes/TP4/src/mat.c
Ir
-- line 17 -----
.     #define INITRANDOMRANGE 10
.     // Macro que realiza swap sem variável auxiliar
.     #define ELEM_SWAP(x, y) (x = y, y = x, x = y)
.
.     void criaMatriz(mat_tipo *mat, int tx, int ty, int id)
.     // Descrição: cria matriz com dimensões tx X ty
.     // Entrada: mat, tx, ty, id
.     // Saída: mat
28 (0.0%) {
.     // verifica se os valores de tx e ty são válidos
8 (0.0%) erroAssert(tx>0, "Dimensão nula");
8 (0.0%) erroAssert(ty>0, "Dimensão nula");
8 (0.0%) erroAssert(tx<=MAXTAM, "Dimensão maior que permitido");
8 (0.0%) erroAssert(ty<=MAXTAM, "Dimensão maior que permitido");
.
.     // inicializa as dimensões da matriz
12 (0.0%) mat->tamx = tx;
12 (0.0%) mat->tamy = ty;
.     // inicializa o identificador da matriz, para rastreamento
12 (0.0%) mat->id = id;
12 (0.0%) }

.     void inicializaMatrizNula(mat_tipo *mat)
.     // Descrição: inicializa mat com valores nulos
.     // Entrada: mat
.     // Saída: mat
12 (0.0%) {
.         int i, j;
.         // inicializa todos os elementos da matriz com 0, por segurança
6,016 (0.0%) for (i=0; i<MAXTAM; i++){
3,008,000 (3.8%)     for(j=0; j<MAXTAM; j++) {
```

```
        // Saída: out
.         {
.             int i,j;
.
-- line 102 -----
-- line 135 -----
.         }
.     }
.
.     void multiplicaMatrizes(mat_tipo *a, mat_tipo *b, mat_tipo *c)
.     // Descrição: multiplica as matrizes a e b e armazena o resultado em c
.     // Entrada: a,b
.     // Saída: c
6 (0.0%) {
.         int i,j,k;
.         // verifica a compatibilidade das dimensões
6 (0.0%) erroAssert(a->tamy==b->tamx, "Dimensões incompatíveis");
.
.         // cria e inicializa a matriz c
9 (0.0%) criaMatriz(c,a->tamx, b->tamy,c->id);
3 (0.0%) inicializaMatrizNula(c);
.
.         // realiza a multiplicação de matrizes
506 (0.0%) for (i=0; i<c->tamx;i++){
50,000 (0.1%)     for (j=0; j<c->tamy;j++){
5,060,000 (6.4%)         for (k=0; k<a->tamy;k++){
34,000,000 (42.9%)             c->m[i][j] += a->m[i][k]*b->m[k][j];
.         }
.     }
4 (0.0%) }

.     void transpõeMatriz(mat_tipo *a)
.     // Descrição: transpõe a matriz a
.     // Entrada: a
```

## 5. Callgrind:

```
$ valgrind --tool=callgrind bin/matop -m -x 100 -y 100
==43245== Callgrind, a call-graph generating cache profiler
==43245== Copyright (C) 2002-2017, and GNU GPL'd, by Josef Weidendorfer et al.
==43245== Using Valgrind-3.21.0 and LibVEX; rerun with -h for copyright info
==43245== Command: bin/matop -m -x 100 -y 100
==43245==
==43245== For interactive control, run 'callgrind_control -h'.
      0   1   2   3   4   5   6   7   8   9   10   11   12
  13   14   15   16   17   18   19   20   21   22   23   24   25   26
  27   28   29   30   31   32   33   34   35   36   37   38   39   40
 41   42   43   44   45   46   47   48   49   50   51   52   53   54   55
 56   57   58   59   60   61   62   63   64   65   66   67   68   69
 70   71   72   73   74   75   76   77   78   79   80   81   82   83
 84   85   86   87   88   89   90   91   92   93   94   95   96   97
 98   99
 0 2357.71 2451.76 2134.93 2260.74 2201.76 2534.09 2437.88 2482.72 2177.95 2287.96 2240.65 2765.53 2352.69 2
177.45 2394.90 2543.28 2090.36 2514.53 2597.81 2404.50 2243.11 2219.89 2574.92 2167.88 2127.55 2375.62 2304.14 249
3.34 2213.35 2122.63 2429.13 2490.08 2342.46 2500.33 2530.21 2441.39 2384.55 2752.00 2650.02 2291.87 2350.23 2444.
94 2455.21 2527.99 2112.30 2294.27 2159.13 2357.05 2386.82 2466.82 2291.34 2532.72 2385.20 2196.05 2581.32 2295.20
2687.42 2666.45 2052.49 2373.29 2288.00 2111.87 2243.64 2377.55 2281.14 2187.01 2468.00 2829.27 2234.10 2479.81
2525.95 2320.04 2271.66 2356.12 2324.44 2230.48 2159.46 2636.47 2545.32 2250.34 2258.86 2479.82 2555.97 2287.38 24
17.32 2129.45 2693.14 2185.52 2386.01 2292.79 2377.77 2233.83 2368.08 2310.77 2538.26 2323.00 2001.50 2436.87 2537

57 2483.85 2693.74 2305.77 2569.72 2587.69 2758.67 2549.18 2595.36 2523.52 2613.32 2427.19 2421.53 2876.61 2539.46
2734.71 2736.05 2389.45 2765.96 2579.76 2300.36 2261.08 2677.39 2511.12 2414.81 2629.07 2959.19 2596.28 2697.91
2622.19 2573.06 2556.31 2625.35 2595.00 2378.04 2386.33 2925.50 2702.95 2548.29 2448.71 2576.48 2453.93 2371.03 25
17.81 2285.65 2767.88 2282.92 2586.02 2467.92 2569.74 2712.13 2761.72 2469.91 2470.01 2614.04 2226.64 2522.04 2685
.25 2264.80
==68233==

==68233== Events : Ir Dr Dw I1mr D1mr D1mw I1mr DLmr DLmw
==68233== Collected : 79180682 28933764 8198613 1543 140442 128403 1514 1115 94304
==68233==

==68233== I refs: 79,180,682
==68233== I1 misses: 1,543
==68233== L1 misses: 1,514
==68233== I1 miss rate: 0.00%
==68233== L1 miss rate: 0.00%
==68233==

==68233== D refs: 37,132,377 (28,933,764 rd + 8,198,613 wr)
==68233== D1 misses: 268,845 ( 140,442 rd + 128,403 wr)
==68233== L1d misses: 95,419 ( 1,115 rd + 94,304 wr)
==68233== D1 miss rate: 0.7% ( 0.5% + 1.6% )
==68233== L1d miss rate: 0.3% ( 0.0% + 1.2% )
==68233==

==68233== LL refs: 270,388 ( 141,985 rd + 128,403 wr)
==68233== LL misses: 96,933 ( 2,629 rd + 94,304 wr)
==68233== LL miss rate: 0.1% ( 0.0% + 1.2% )


```

## flag -cache-yes=yes

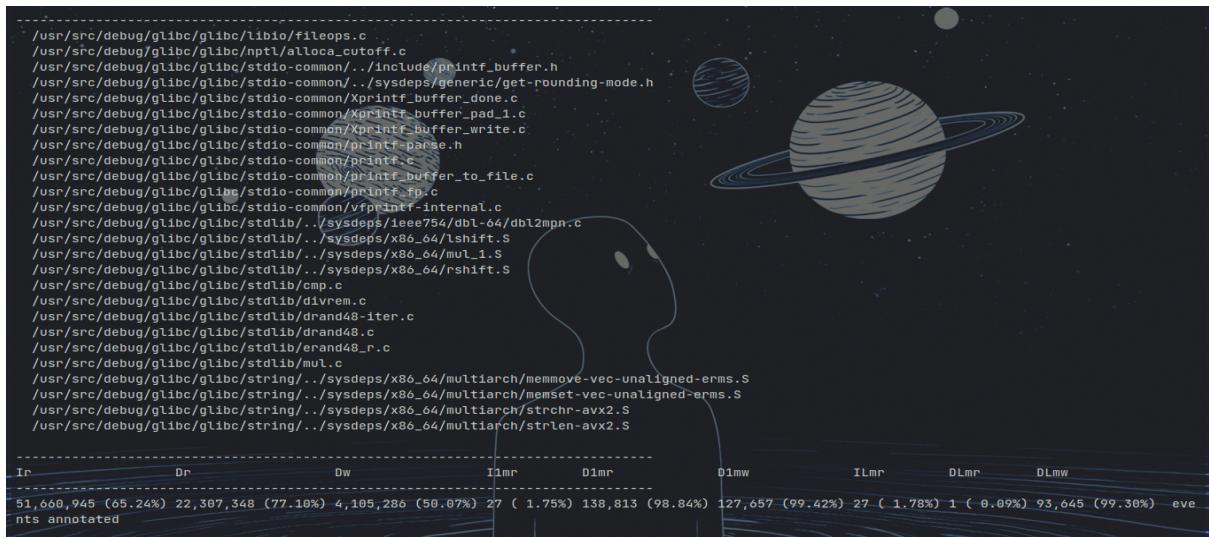
```
-----Profile data file 'callgrind.out.68233' (creator: callgrind-3.21.0)-----
-----I1 cache: 32768 B, 64 B, 8-way associative
-----D1 cache: 32768 B, 64 B, 8-way associative
-----LL cache: 6291456 B, 64 B, 12-way associative
-----Timerange: Basic block 0 - 754525
-----Trigger: Program termination
-----Profiled target: bin/matop -m -x 100 -y 100 (PID 68233, part 1)
-----Events recorded: Ir Dr Dw I1mr D1mr I1mw D1mr DLmr DLmw
-----Events shown: Ir Dr Dw I1mr D1mr I1mw D1mr DLmr DLmw
-----Event sort order: Ir Dr Dw I1mr D1mr I1mw D1mr DLmr DLmw
-----Thresholds: 99 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
-----Include dirs:
-----User annotated:
-----Auto-annotation: on
-----
```



```
-----Ir Dr Dw I1mr D1mr D1mw I1mr DLmr DLmw f
-----79,180,684 (100.0%) 28,933,764 (100.0%) 8,198,613 (100.0%) 1,543 (100.0%) 140,442 (100.0%) 128,403 (100.0%) 1,514 (100.0%) 1,115 (100.0%) 94,304 (100.0%) PROGRAM TOTALS
-----
```

```
-----Ir Dr Dw I1mr D1mr D1mw I1mr DLmr DLmw f
-----39,111,134 (49.39%) 18,860,617 (62.42%) 2,020,287 (24.64%) 7 ( 0.45%) 137,903 (97.91%) 1 ( 0.00%) 7 ( 0.46%) . . . s
-----rc/mat.c:multiplicaMatrices [/home/iaigo/Documents/school/ufmg/3.Semestre/ED/codes/TP4/bin/matop]
-----12,014,044 (15.17%) 4,004,012 (13.84%) 2,004,012 (24.44%) 0 0 125,000 (97.35%) 0 0 93,043 (99.30%) s
-----rc/mat.c:inicializaMatrizNula [/home/iaigo/Documents/school/ufmg/3.Semestre/ED/codes/TP4/bin/matop]
-----6,725,799 ( 8.40%) 1,390,412 ( 4.81%) 755,554 ( 9.22%) 67 ( 4.34%) 9 ( 0.01%) 7 ( 0.01%) 67 ( 4.43%) 9 ( 0.81%) 7 ( 0.01%) /
-----usr/src/debug/glibc/glibc/stdio-common/printf_fp.c:_printf_fp_buffer_1.isra.0 [/usr/lib/libc.so.6]
-----3,607,965 ( 4.56%) 1,317,186 ( 4.55%) 547,186 ( 6.67%) 3 ( 0.19%) 0 0 3 ( 0.28%) . . /
```

```
-----Ir Dr Dw I1mr D1mr D1mw I1mr DLmr DLmw f
-----usr/src/debug/glibc/glibc/stdio-common/printf_fp.c:hack_digit [/usr/lib/libc.so.6]
```



## flag -cache-yes=no

```
$ callgrind_annotate callgrind.out.43245
Profile data file 'callgrind.out.43245' (creator: callgrind-3.21.0)

I1 cache:
D1 cache:
LL cache:
Timerange: Basic block 0 - 7544525
Trigger: Program termination
Profiled target: bin/matop -m -x 100 -y 100 (PID 43245, part 1)
Events recorded: Ir
Events shown: Ir
Event sort order: Ir
Thresholds: 99
Include dirs:
User annotated:
Auto-annotation: on

Ir
79,180,682 (100.0%) PROGRAM TOTALS

Ir      file:function
39,111,134 (49.39%) src/mat.c:multiplicaMatrices [/home/iaigo/Documents/school/ufmg/3_Semestre/ED/codes/TP4/bin/matop]
12,814,944 (15.17%) src/mat.c:inicializaMatrizNula [/home/iaigo/Documents/school/ufmg/3_Semestre/ED/codes/TP4/bin/matop]
6,725,799 ( 8.49%) /usr/src/debug/glibc/glibc/stdio-common/printf_fp.c:__printf_fp_buffer_1.isra.0 [/usr/lib/libc.so.6]
3,607,965 ( 4.56%) /usr/src/debug/glibc/glibc/stdio-common/printf_fp.c:hack_digit [/usr/lib/libc.so.6]
3,166,098 ( 4.00%) /usr/src/debug/glibc/glibc/stdlib/divrem.c:__mpn_divrem [/usr/lib/libc.so.6]
2,184,321 ( 2.76%) /usr/src/debug/glibc/glibc/stdio-common/vprintf_internal.c:__printf_buffer [/usr/lib/libc.so.6]
1,800,000 ( 2.27%) /usr/src/debug/glibc/glibc/stdlib/../sysdeps/x86_64/mul_1.S:__mpn_mul_1 [/usr/lib/libc.so.6]
948,056 ( 1.20%) /usr/src/debug/glibc/glibc/libio/fileops.c:_IO_file_xsputn@GLIBC_2.2.5 [/usr/lib/libc.so.6]
877,270 ( 1.11%) /usr/src/debug/glibc/glibc/stdio-common/Xprintf_buffer_write.c:__printf_buffer_write [/usr/lib/libc.so.6]
```

```
void inicializaMatrizNula(mat_tipo * mat)
// Descricao: inicializa mat com valores nulos
// Entrada: mat
// Saída: mat
12 ( 0.00% ) {
    . . .
    int i, j;
    // inicializa todos os elementos da matriz com 0, por segurança
    6,010 ( 0.01% ) for (i=0; i<MAXTAM; i++){
        3,008,000 ( 3.80% ) for (j=0; j<MAXTAM; j++){
            9,000,000 ( 11.37% ) mat->m[i][j] = 0;
            .
        }
    }
16 ( 0.00% ) }

void inicializaMatrizAleatoria(mat_tipo * mat)
// Descricao: inicializa mat com valores aleatórios
// Entrada: mat
// Saída: mat
8 ( 0.00% ) {
    . . .
    int i, j;
    // inicializa a matriz com valores nulos, por segurança
    6 ( 0.00% ) inicializaMatrizNula(mat);
    6,007,022 ( 7.59% ) => src/mat.c:inicializaMatrizNula (2x)
        // inicializa a parte alocada da matriz com valores aleatórios
        for (i=0; i<mat->tamx; i++){
            1,012 ( 0.00% ) for (j=0; j<mat->tamy; j++){
                101,200 ( 0.13% ) mat->m[i][j] = drand48() * INITRANDOMRANGE;
                249,004 ( 0.36% ) .
                1,320,003 ( 1.67% ) => /usr/src/debug/glibc/glibc/stdlib/drnd48.c:drand48 (20,000x)
                725 ( 0.00% ) => /usr/src/debug/glibc/glibc/elf/..../sysdeps/x86_64/dl-trampoline.h:_dl_runtime_resolve_xsave (1x)
                .
            }
        }
    8 ( 0.00% ) }

void imprimeMatriz(mat_tipo * mat)
// Descricao: imprime a matriz com a identificação de linhas e colunas
// Entrada: mat

```

```
void multiplicaMatrizes(mat_tipo *a, mat_tipo *b, mat_tipo *c)
    // Descrição: multiplica as matrizes a e b e armazena o resultado em c
    // Entrada: a,b
    // Saída: c
6 ( 0.00%) {
    .           int i,j,k;
    .           // verifica a compatibilidade das dimensões
6 ( 0.00%)   erroAssert(a->tamy==b->tamx,"Dimensões incompatíveis");
    .
    .           // cria e inicializa a matriz c
9 ( 0.00%)   criaMatriz(c,a->tamx, b->tamy,c->id);
27 ( 0.00%) => src/mat.c:criaMatriz (1x)
    3 ( 0.00%)   inicializaMatrizNula(c);
3,003,511 ( 3.79%) => src/mat.c:inicializaMatrizNula (1x)

    .
    .           // realiza a multiplicação de matrizes
506 ( 0.00%)   for (i=0; i<c->tamx;i++){
    50,600 ( 0.06%)     for (j=0; j<c->tamy;j++){
    5,060,000 ( 6.39%)       for (k=0; k<a->tamy;k++){
    34,000,000 ( 42.94%)         c->m[1][j] += a->m[1][k]*b->m[k][j];
    .
    .
    .
    4 ( 0.00%) }
```

**6. 1)** De forma geral, o programa se comportou bem com relação à memória. Ambos I1 e L1 misses tiveram uma taxa muito próxima de zero, o que é uma indicação bem positiva.

Com relação aos misses de dados, os valores são bons, mas não exatamente excelentes com as taxas de 0.3% e 0.7%, ou seja, é considerada aceitável, mas podia ser melhor otimizado.

Já as taxas de misses de último nível estão com uma taxa ótima de 0.1%, relatando que a grande maioria dos dados está sendo atendido pela cache de dados de último nível.

Em resumo, considerando a quantidade massiva de instruções executadas(79,263,288), mesmo com muitos acessos à quantidade de memória, o programa se comportou bem e manteve taxas baixas de misses de cache, instruções e de dados. Apesar de não ser a única métrica necessária para avaliar inteiramente o acesso de memória, é um ótimo indicativo.

**2)** A estrutura de dados a ser caracterizada para melhor entendimento é a struct “mat”, pois o restante do código inteiro depende de sua implementação. A estrutura deve ser analisada e bem compreendida, pois matrizes grandes podem ser especialmente densas e consumir uma parte significativa da memória se não implementadas de forma otimizada. Por isso, é necessário um bom entendimento da estrutura “mat”.

**3)** Os pontos de segmento crítico do programa estão localizados nas funções de inicialização e operação de matrizes, além de liberação da memória.

Com relação à matriz de inicialização, é perceptível, através do uso das ferramentas callgrind e cachegrind, uma porcentagem significativa de instruções executadas na linha onde a posição da matriz sendo instanciada é acessada e assinalada como “0”, por isso a necessidade de um olhar mais crítico na hora de instanciar as matrizes e não gerar nenhum acesso errado ou desperdício de memória.

Nas funções de operações de matrizes, principalmente para matrizes muito grandes, a maior porcentagem de instruções e acessos de memória ficam para as linhas dessas funções que geram um resultado e escrevem na matriz de retorno. Além disso, é crucial avaliar as iterações utilizadas para acessar a estrutura de dados, pois dependendo de como são implementados podem gerar um impacto significativo no uso de memória.

Por último, deve-se tomar cuidado ao destruir a estrutura de dados utilizada no programa para ter certeza que qualquer alocação é liberada adequadamente.