Universidade Federal do Rio Grande do Norte
Departamento de Engenharia da Computação e Automação
DCA3703 - Programação Paralela
Tarefa 1 - Memória Cache
Aluno: Daniel Bruno Trindade da Silva

Introdução:

Nesta tarefa, exploramos a multiplicação de matriz por vetor (MxV), com o objetivo de analisar como dois diferentes padrões de acesso aos valores da matriz (por linhas e por colunas) impactam no tempo de execução e o motivo pelo qual as execuções diferem.

A tarefa consiste em implementar duas versões do algoritmo em C, uma que percorre a matriz por linhas (laço externo sobre linhas, interno sobre colunas) e outra que acessa os elementos por colunas (laço externo sobre colunas, interno sobre linhas). Ao executar ambas as versões faremos a medição de tempo, com o fim de comparar o desempenho das duas abordagens em matrizes de diferentes tamanhos, identificando a partir de qual dimensão os tempos de execução passam a divergir significativamente.

Implementação:

Para testar as duas abordagens e obter seus resultados em um único código, implementamos uma função para cada uma delas: `mat_vec_row`, que realiza o acesso aos elementos da matriz por linhas, e `mat_vet_col`, que acessa os elementos por colunas. Com o código estruturado dessa forma, podemos testar ambas as abordagens em uma única execução.

Faremos a chamada de ambas as funções em um laço de repetição com o fim de fazer testes aumentando o numero de elementos da matriz e do vetor em cada repetição para identificar a partir de qual dimensão começamos a ter uma grande divergência no tempo de execução.

Para execução da análise, realizamos a multiplicação da matriz com tamanho $n \times n$ por um vetor também de tamanho n com n assumindo os valores de 100, 1000, 10000, 50000 e 100000. Em cada uma das execuções são medidos os tempos de tomados por cada uma das abordagens propostas e entregues em segundos.

Código:

O código utiliza a biblioteca `stdlib.h` para alocação de memória e a `time.h` para aferição do tempo de execução. O código ficou como se segue:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Função para medir o tempo
double get_time(clock_t start, clock_t end) {
 return (double)(end - start) / CLOCKS_PER_SEC;
}

// Multiplicacao MxV com acesso por linhas
void mat_vec_row(double** mat, double* vec, double* res, int n) {
 for (int i = 0; i < n; i++) {
```

```
  res[i] = 0.0;
  for (int j = 0; j < n; j++) {
   res[i] += mat[i][j] * vec[j];
  }
 }
}

// Multiplicacao MxV com acesso por colunas
void mat_vec_col(double** mat, double* vec, double* res, int n) {
 for (int j = 0; j < n; j++) {
  for (int i = 0; i < n; i++) {
   res[i] += mat[i][j] * vec[j];
  }
 }
}

int main() {
 int sizes[] = {100, 500, 1000, 2000, 3000};
 int num_tests = 5;

 for (int t = 0; t < num_tests; t++) {
  int n = sizes[t];
  printf("Tamanho da matriz: %d\n", n);

  // Alocação de matriz e vetores
  double** mat = (double**) malloc(n * sizeof(double*));
  double* vec = (double*) malloc(n * sizeof(double));
  double* res_row = (double*) calloc(n, sizeof(double));
  double* res_col = (double*) calloc(n, sizeof(double));

  for (int i = 0; i < n; i++) {
   mat[i] = (double*) malloc(n * sizeof(double));
   for (int j = 0; j < n; j++) {
    mat[i][j] = (double)rand() / RAND_MAX;
   }
   vec[i] = (double)rand() / RAND_MAX;
  }

  // Tempo para acesso por linhas
  clock_t start = clock();
  mat_vec_row(mat, vec, res_row, n);
  clock_t end = clock();
  printf("Tempo (acesso por linhas): %.4f s\n", get_time(start, end));

  // Tempo para acesso por colunas
  start = clock();
  mat_vec_col(mat, vec, res_col, n);
  end = clock();
  printf("Tempo (acesso por colunas): %.4f s\n\n", get_time(start, end));
```

```
  // Liberação de memória
  for (int i = 0; i < n; i++) {
   free(mat[i]);
  }
  free(mat);
  free(vec);
  free(res_row);
  free(res_col);
 }

 return 0;
}
```

Resultados:
Após execução do código obtivemos os seguintes resultados:

valor de n	**Tempo de execução por linhas**	**Tempo de execução por colunas**
100	0,0002s	0,0002s
500	0,0045s	0,0085s
1.000	0,0094s	0,0215s
5.000	0,0940s	0,3941s
10.000	0,3517s	2,1709s
15.000	0,7934s	6,1648s

Table 1: Tempo de Execução par cada valor de n

Como é possível observar, o tempo de execução na abordagem por coluna, se torna muito superior ao da abordagem por linhas. Com *n=500* o tempo da versão por colunas já quase o dobro do tempo da por linhas. No teste com *n=15.000* o tempo usado pelo função que percorre as colunas é quase 8 vezes o tempo usado pela execução por linhas. Podemos observar melhor com o seguinte gráfico: