RELATÓRIO COMPUTAÇÃO PARALELA

Tema: Você sabe multiplicar uma matriz?

Grupo

- Lucas Zanini da Silva - 10417361

Sobre o laboratório

- Foi orientado para que seja realizada a implementação de um algoritmo de multiplicação de matrizes por meio de ordem de linha e utilizando o método de blocagem. Além disso, foi requisitada a compilação do código sem otimizações (-O0) e com o máximo de otimização por parte de compilador (-O3).
- O laboratório foi realizado na IDE CLION utilizando o terminal WSL para utilizar o utilitário valgrind.
- Link do repositório do experimento: https://github.com/LucasZanini096/COMPUTACAO_PARALELA/tree/master/Lab03

Definição de funções

Foi utilizado para haver a definição de todas as funções necessárias para o experimento, por meio de um arquivo com extensão .h.

```h
#ifndef MATRIX_H
#define MATRIX_H

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

#define NUM_LINES_A 1000
#define NUM_COLS_A 1000

#define NUM_LINES_B 1000
#define NUM_COLS_B 1000

void showMatrix(int nLines, int nColumns, int matrix[nLines][nColumns]);

void constructMatrix(int nLines, int nColumns, int matrix[nLines][nColumns], char operation);

int multiMatrixRow(int matrixA[NUM_LINES_A][NUM_COLS_A], int matrixB[NUM_LINES_B][NUM_COLS_B], int matrizC[NUM_LINES_A][NUM_COLS_B]);

int multiMatrixBlock(int matrixA[NUM_LINES_A][NUM_COLS_A], int matrixB[NUM_LINES_B][NUM_COLS_B], int matrizC[NUM_LINES_A][NUM_COLS_B]);

#endif
```

Multiplicação de matrizes por ordem de linha

Definição a da função para a multiplicação de matrizes

```c
#include "../include/matrix.h"

int multiMatrixRow(int matrixA[NUM_LINES_A][NUM_COLS_A], int matrixB[NUM_LINES_B][NUM_COLS_B], int matrizC[NUM_LINES_A][NUM_COLS_B]) {

    if (NUM_COLS_A != NUM_COLS_B) return -1;

     for (int i = 0; i < NUM_LINES_A; i++) {
        for (int j = 0; j < NUM_COLS_B; j++) {
            int sum = 0;
            for (int k = 0; k < NUM_COLS_A; k++) {
                sum += matrixA[i][k] * matrixB[k][j];
            }
            matrizC[i][j] = sum;
        }
     }

    return 0;
}
```

Resultado da execução da função com -O0

```
lucas_zanini@Inspiron15:/mnt/c/Mackenzie/Quinto Semestre/Computacao_Paralela/Lab03$ valgrind --tool=cachegrind ./multiRow_O0
==21405== Cachegrind, a high-precision tracing profiler
==21405== Copyright (C) 2002-2024, and GNU GPL'd, by Nicholas Nethercote et al.
==21405== Using Valgrind-3.24.0 and LibVEX; rerun with -h for copyright info
==21405== Command: ./multiRow_O0
==21405==
Iniciando a multiRow...
A operação foi realizada com sucesso!

Tempo de execução: 56888.426000 ms==21405==
==21405== I refs:        21,059,175,464
```

Resultado da execução da função com -O3

```
lucas_zanini@Inspiron15:/mnt/c/Mackenzie/Quinto Semestre/Computacao_Paralela/Lab03$ valgrind --tool=cachegrind ./multiRow_O3
==22895== Cachegrind, a high-precision tracing profiler
==22895== Copyright (C) 2002-2024, and GNU GPL'd, by Nicholas Nethercote et al.
==22895== Using Valgrind-3.24.0 and LibVEX; rerun with -h for copyright info
==22895== Command: ./multiRow_O3
==22895==
Iniciando a multiRow...
A operação foi realizada com sucesso!

Tempo de execução: 13538.253000 ms==22895==
==22895== I refs:        5,271,178,563
```

Multiplicação de matrizes por blocagem

Definição da função para a multiplicação das matrizes por blocagem

```c
#include "../include/matrix.h"


int multiMatrixBlock(int matrixA[NUM_LINES_A][NUM_COLS_A], int matrixB[NUM_LINES_B][NUM_COLS_B], int matrizC[NUM_LINES_A][NUM_COLS_B]) {

    if (NUM_COLS_A != NUM_LINES_B) return -1;

    int i, j, k, iInner, kInner,jInner;
    int blockSize = 10;

    for (i = 0; i < NUM_LINES_A; i += blockSize) {
        for (j = 0; j < NUM_COLS_B; j += blockSize) {
            for (k = 0; k < NUM_COLS_A; k += blockSize) {
                for (iInner = i; iInner < i + blockSize && iInner < NUM_LINES_A; iInner++) {
                    for (jInner = j; jInner < j + blockSize && jInner < NUM_COLS_B; jInner++) {
                        for (kInner = k; kInner < k + blockSize && kInner < NUM_COLS_A; kInner++) {
                            matrizC[iInner][jInner] += matrixA[iInner][kInner] * matrixB[kInner][jInner];
                        }
                    }
                }
            }
        }
    }

    return 0;
}
```

Resultado da execução da função com -O0

```
lucas_zanini@Inspiron15:/mnt/c/Mackenzie/Quinto Semestre/Computacao_Paralela/Lab03$ valgrind --tool=cachegrind ./multiBlock_O0
==26178== Cachegrind, a high-precision tracing profiler
==26178== Copyright (C) 2002-2024, and GNU GPL'd, by Nicholas Nethercote et al.
==26178== Using Valgrind-3.24.0 and LibVEX; rerun with -h for copyright info
==26178== Command: ./multiBlock_O0
==26178==
Iniciando o multiBlock...
A operação foi realizada com sucesso!

Tempo de execução: 141962.450000 ms==26178==
==26178== I refs:        44,814,249,629
```

Resultado da execução da função com -O3

```
lucas_zanini@Inspiron15:/mnt/c/Mackenzie/Quinto Semestre/Computacao_Paralela/Lab03$ valgrind --tool=cachegrind ./multiBlock_O3
==29035== Cachegrind, a high-precision tracing profiler
==29035== Copyright (C) 2002-2024, and GNU GPL'd, by Nicholas Nethercote et al.
==29035== Using Valgrind-3.24.0 and LibVEX; rerun with -h for copyright info
==29035== Command: ./multiBlock_O3
==29035==
Iniciando o multiBlock...
A operação foi realizada com sucesso!

Tempo de execução: 12136.669000 ms==29035==
==29035== I refs:        5,901,392,781
```