Multiplicação de matrizes em C++

Edson Luiz Cardoso Ohira – 10419663 – 04P

EX1-

Multiplicação_linhas.cpp:

```cpp
#include <iostream>

#include <vector>
#include <cstdlib>
#include <ctime>

using namespace std;

const int LINHAS = 500, COLUNAS = 500, X = 500;

void gerarMatriz(vector<vector<int>>& matriz, int linhas, int colunas) {
    for (int i = 0; i < linhas; ++i) {
        for (int j = 0; j < colunas; ++j) {
            matriz[i][j] = rand() % 10;
        }
    }
}

void multiplicarPorLinha(const vector<vector<int>>& A, const
vector<vector<int>>& B, vector<vector<int>>& C) {
    for (int i = 0; i < LINHAS; ++i) {
        for (int j = 0; j < X; ++j) {
            C[i][j] = 0;
            for (int k = 0; k < COLUNAS; ++k) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}

int main() {
    srand(time(0));

    vector<vector<int>> matrizA(LINHAS, vector<int>(COLUNAS));
    vector<vector<int>> matrizB(COLUNAS, vector<int>(X));
    vector<vector<int>> matrizC(LINHAS, vector<int>(X, 0));

    gerarMatriz(matrizA, LINHAS, COLUNAS);
    gerarMatriz(matrizB, COLUNAS, X);
```

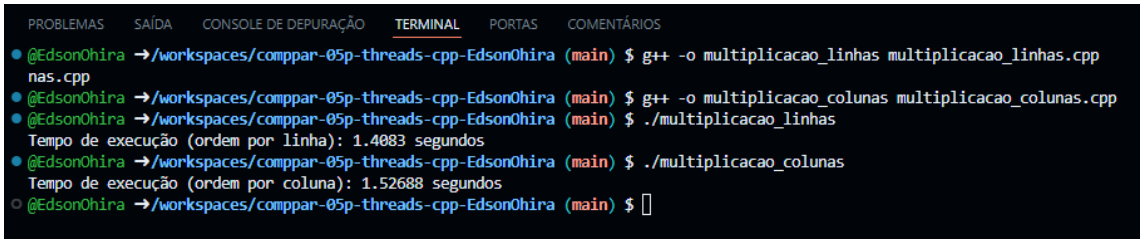
```
    clock_t inicio = clock();
    multiplicarPorLinha(matrizA, matrizB, matrizC);
    clock_t fim = clock();

    cout << "Tempo de execução (ordem por linha): "
         << (double)(fim - inicio) / CLOCKS_PER_SEC << " segundos" <<
endl;

    return 0;
}
```

Multiplicação_colunas.cpp:

```
#include <iostream>

#include <vector>
#include <cstdlib>
#include <ctime>

using namespace std;

const int LINHAS = 500, COLUNAS = 500, X = 500;

void gerarMatriz(vector<vector<int>>& matriz, int linhas, int colunas) {
    for (int i = 0; i < linhas; ++i) {
        for (int j = 0; j < colunas; ++j) {
            matriz[i][j] = rand() % 10;
        }
    }
}

void multiplicarPorColuna(const vector<vector<int>>& A, const
vector<vector<int>>& B, vector<vector<int>>& C) {
    for (int j = 0; j < X; ++j) {
        for (int i = 0; i < LINHAS; ++i) {
            C[i][j] = 0;
        }
        for (int k = 0; k < COLUNAS; ++k) {
            for (int i = 0; i < LINHAS; ++i) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}

int main() {
    srand(time(0));
```

```
    vector<vector<int>> matrizA(LINHAS, vector<int>(COLUNAS));
    vector<vector<int>> matrizB(COLUNAS, vector<int>(X));
    vector<vector<int>> matrizC(LINHAS, vector<int>(X, 0));

    gerarMatriz(matrizA, LINHAS, COLUNAS);
    gerarMatriz(matrizB, COLUNAS, X);

    clock_t inicio = clock();
    multiplicarPorColuna(matrizA, matrizB, matrizC);
    clock_t fim = clock();

    cout << "Tempo de execução (ordem por coluna): "
         << (double)(fim - inicio) / CLOCKS_PER_SEC << " segundos" <<
endl;

    return 0;
}
```

Compilação e execução dos executáveis:



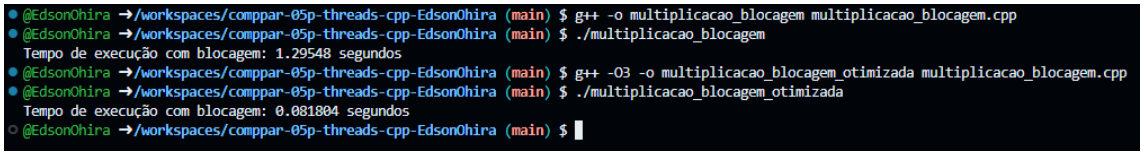
EX2-

Multiplicação_blocagem.cpp:

```
#include <iostream>

#include <vector>
#include <cstdlib>
#include <ctime>

using namespace std;

const int LINHAS = 500, COLUNAS = 500, X = 500;
const int TAMANHO_BLOCO = 50;

void gerarMatriz(vector<vector<int>>& matriz, int linhas, int colunas) {
    for (int i = 0; i < linhas; ++i) {
```

```
        for (int j = 0; j < colunas; ++j) {
            matriz[i][j] = rand() % 10;
        }
    }
}

void multiplicarComBlocagem(const vector<vector<int>>& A, const
vector<vector<int>>& B, vector<vector<int>>& C) {
    for (int i0 = 0; i0 < LINHAS; i0 += TAMANHO_BLOCO) {
        for (int j0 = 0; j0 < X; j0 += TAMANHO_BLOCO) {
            for (int k0 = 0; k0 < COLUNAS; k0 += TAMANHO_BLOCO) {
                for (int i = i0; i < min(i0 + TAMANHO_BLOCO, LINHAS);
++i) {
                    for (int j = j0; j < min(j0 + TAMANHO_BLOCO, X); ++j)
{
                        int soma = 0;
                        for (int k = k0; k < min(k0 + TAMANHO_BLOCO,
COLUNAS); ++k) {
                            soma += A[i][k] * B[k][j];
                        }
                        C[i][j] += soma;
                    }
                }
            }
        }
    }
}

int main() {
    srand(time(0));

    vector<vector<int>> matrizA(LINHAS, vector<int>(COLUNAS));
    vector<vector<int>> matrizB(COLUNAS, vector<int>(X));
    vector<vector<int>> matrizC(LINHAS, vector<int>(X, 0));

    gerarMatriz(matrizA, LINHAS, COLUNAS);
    gerarMatriz(matrizB, COLUNAS, X);

    clock_t inicio = clock();
    multiplicarComBlocagem(matrizA, matrizB, matrizC);
    clock_t fim = clock();

    cout << "Tempo de execução com blocagem: "
         << (double)(fim - inicio) / CLOCKS_PER_SEC << " segundos" <<
endl;

    return 0;
}
```

Compilação e execução do executável:



EX3- Utilização do valgrind

Para utilizar o valgrind fiz o uso de uma máquina virtual, limitando a arquitetura do cache do processador real. Acredito que por esse motivo não há muitas informações ao executar o código.

```
edson@edson-VirtualBox:~/Downloads$ valgrind --tool=cachegrind ./multiplicacao_colunas
==9887== Cachegrind, a high-precision tracing profiler
==9887== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==9887== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==9887== Command: ./multiplicacao_colunas
==9887==
Tempo de execução (ordem por coluna): 35.0768 segundos
==9887==
==9887== I refs:        15,826,566,583
edson@edson-VirtualBox:~/Downloads$ valgrind --tool=cachegrind ./multiplicacao_linhas
==9970== Cachegrind, a high-precision tracing profiler
==9970== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==9970== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==9970== Command: ./multiplicacao_linhas
==9970==
Tempo de execução (ordem por linha): 35.1504 segundos
==9970==
==9970== I refs:        15,825,814,529
edson@edson-VirtualBox:~/Downloads$ valgrind --tool=cachegrind ./multiplicacao_blocagem
==10105== Cachegrind, a high-precision tracing profiler
==10105== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==10105== Using Valgrind-3.22.0 and LibVEX; rerun with -h for copyright info
==10105== Command: ./multiplicacao_blocagem
==10105==
Tempo de execução com blocagem: 30.0288 segundos
==10105==
==10105== I refs:        14,071,994,482
```

- **Tempo de Execução:**
  - **Colunas: 35.0768 s**
  - **Linhas: 35.1504 s**
  - **Blocagem: 30.0288 s**

A multiplicação com blocagem foi a mais eficiente, completando a execução mais rápido (5 segundos a menos que as outras).

- Número de Instruções (I refs):

- **Colunas: 15.8 bilhões**
- **Linhas: 15.8 bilhões**
- **Blocagem: 14.0 bilhões**

A abordagem com blocagem realizou menos instruções, o que confirma sua melhor eficiência.