Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Algoritmos e Estruturas de Dados III

Trabalho Prático 0

Alexandre Torres Pimenta

Belo Horizonte, 29 de Agosto de 2014

Índice

Introdução

O trabalho prático tem como objetivo implementar um programa para multiplicar matrizes de números complexos.

Implementação

Para representar a matriz, foi usado um tipo abstrato de dado, que contém um vetor alocado dinâmicamente, simulando uma matriz na memória primária. Por exemplo, a matriz:

8+6i	10+0i	3+5i
0+3i	8+2i	0+0i
3+1i	10+0i	1+7i

Pode ser representada como vetor, da seguinte maneira:

8+6i	0+3i	3+1i	10+0i	8+2i	10+0i	3+5i	0+0i	1+7i

```
typedef struct Matrix
{
    complexNumber *Body;
    int L;
    int C;
} Matrix;
```

Além do vetor de números complexos, o TAD que representa a matriz possui dois inteiros L e C, que representam a quandidade de linhas e colunas, respectivamente.

```
typedef struct complexNumber
{
    double X;
    double Y;
} complexNumber;
```

Para representar os números complexos, foi criado outro TAD. Ele possui dois doubles, X para representar a parte real do número e Y para representar a parte imaginária.

Funções e procedimentos

Junto com o TAD, foram implementadas as seguintes funções:

complexNumber multiplyComplexNumber(complexNumber A, complexNumber B)

A função retorna o produto entre os números complexos A e B.

complexNumber addComplexNumber(complexNumber A, complexNumber B)

A função retorna a soma dos números complexos A e B.

int matrixOffset(int Colunas, int i, int j)

Essa função é usada para simular a matriz no vetor alocado dinâmicamente. Ela recebe os valores i e j, que seriam usados em uma matriz estática convencional (Matriz[i][j]) e retorna a posição equivalente no vetor.

void printMatrix(Matrix M, FILE *file)

A função percorre as linhas e colunas da matriz, imprimindo seus valores do arquivo de saída.

void printMatrixSTDOUT(Matrix M)

A função percorre as linhas e colunas da matriz, imprimindo seus valores no dispositivo de saída padrão (stdout)

Organização do código e detalhes técnicos

O código está separado em três arquivos. Os arquivos header.h e functions.c são usados para implementar os tipos abstratos de dados e suas operações, e o arquivo main.c implementa o programa principal. O compilador utilizado foi o GNU CCC COMPILER no sistema operacional Microsoft Windows 8.1

Análise de complexidade

complexNumber multiplyComplexNumber(complexNumber A, complexNumber B)

O procedimento realiza 2 atribuições, independente da entrada. Ou seja, sua complexidade é constante, $O(2)$.

complexNumber addComplexNumber(complexNumber A, complexNumber B)

O procedimento realiza 2 atribuições, independente da entrada. Ou seja, sua complexidade é constante, $O(2)$.

int matrixOffset(int Colunas, int i, int j)

O procedimento realiza uma atribuição e retorna esse valor. A complexidade é constante, $O(1)$.

void printMatrix(Matrix M, FILE *file)

O procedimento caminha por todos os registros da matriz, os imprimindo no arquivo de saída. Considerando n o número de registros na matriz, a função tem complexidade $O(n)$.

void printMatrixSTDOUT(Matrix M)

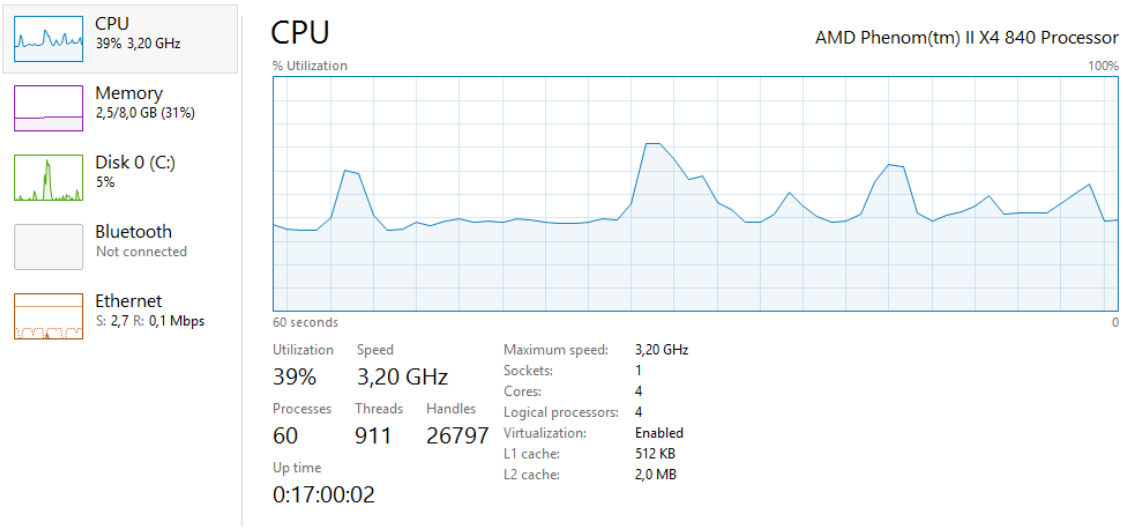
O procedimento caminha por todos os registros da matriz, os impimindo no dispositivo de saída padrão. Considerando n como o número de registros na matriz, a função tem complexidade $O(n)$.

Complexidade do programa principal

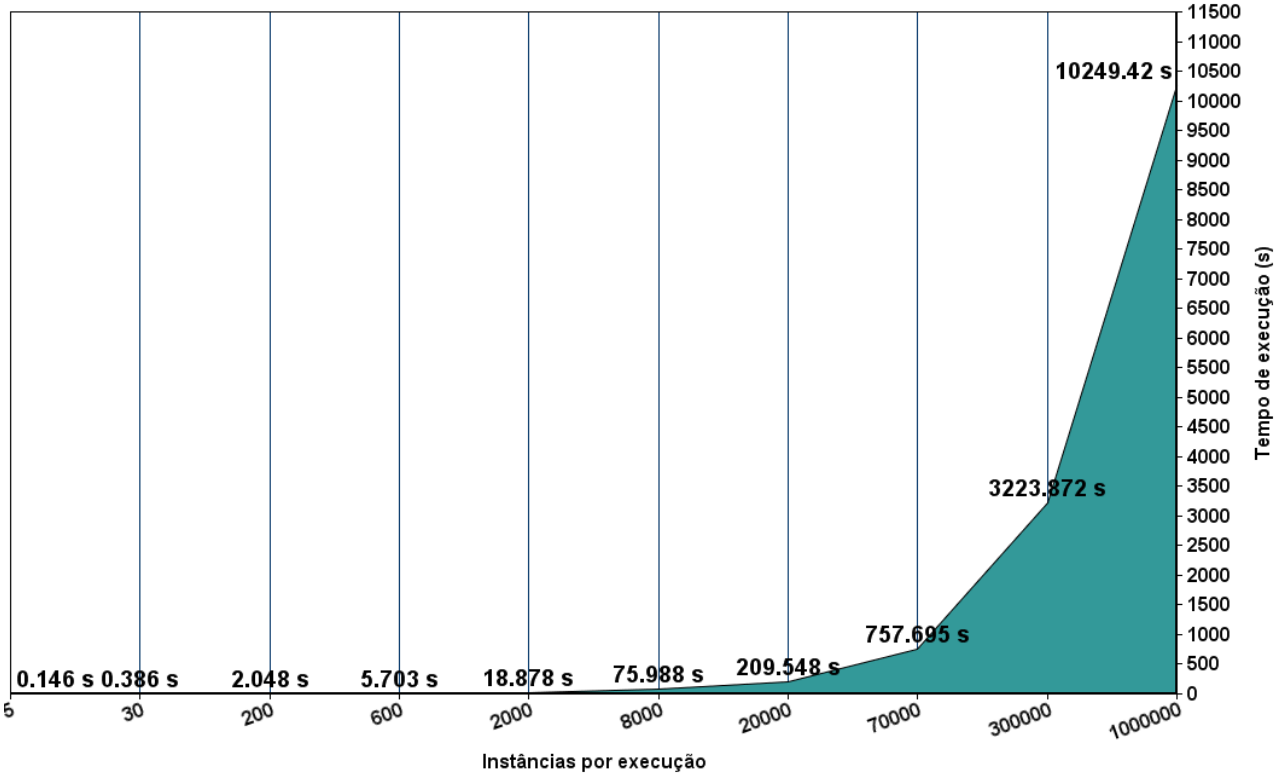
Desconsiderando as atribuições constantes que são assintoticamente insignificantes, o programa lê cada registro da matriz no arquivo de entrada e os passa para a memória principal, multiplica as duas matrizes lidas, imprime a matriz multiplicada, e desaloca a memória para a próxima instância ou término da execução. Durante a multiplicação das matrizes, para cada linha da primeira matriz, o programa acessa todas as colunas da segunda matriz, e para cada coluna da segunda matriz acessada, ele passa por todas as colunas da primeira matriz realizando as somas e multiplicações. Ou seja, dentro de cada loop, ele percorre a matriz uma vez. E como são 3 loops, a complexidade da multiplicação será $O(n^3)$ onde n é o número de elementos das matrizes. Juntando todos esses fatores, a complexidade do programa principal será $O(n^3)$ (do procedimento de multiplicação), + $O(n)$ (da leitura de cada elementos das matrizes para a memória principal) + $O(n)$ do procedimento de impressão. A complexidade final será $O(n^3 + 2n)$.

Testes

Os testes foram realizados usando um AMD Phenom II X4 840, com 8 GB de memória RAM, no sistema operacional Microsoft Windows 8.1. Para realizar os testes, foram utilizados arquivos com matrizes geradas aleatóriamente por outro programa, com o número máximo de linhas e colunas de cada matriz fixados em 10 e com um determinado número de instâncias.

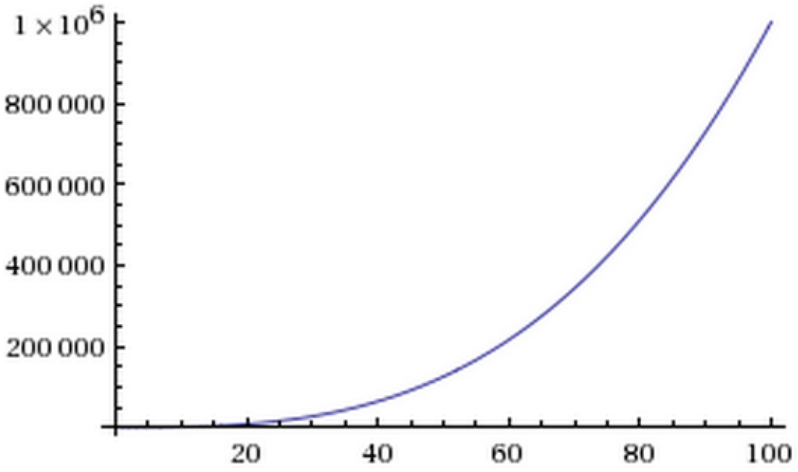


Instâncias por execução	Tempo de execução (s)
1	0.024
5	0.146
30	0.386
200	2.048
600	5.703
2000	18.878
8000	75.988
20000	209.548
70000	757.695
300000	3223.872
1000000	10249.420



Comparando o gráfico de desempenho do programa, com o gráfico da sua função de complexidade $O(n^3 + 2n)$, vemos que os dois gráficos são compatíveis.

Plot:



Conclusão

O trabalho foi essencial para firmar os conceitos de operações com matrizes e operações com números complexos. Durante o desenvolvimento do trabalho, não foram encontradas grandes dificuldades.

Referências

Ziviani, N., Projeto de Algoritmos com Implementações em Pascal e C, 3ª Edição.

http://homepages.dcc.ufmg.br/~rodolfo/aedsi-2-05/arranjo/multiplica-matrizes.html Acessado em 19/08/2014

Anexos

Listagem dos códigos:

- header.h
- functions.c
- main.c
- geradorMatriz.c (Programa utilizado para gerar as matrizes dos testes)