COS284 Practical Assignment 3: Matrix Operations

University of Pretoria
Department of Computer Science

Online Publication: 6 September 2024

Plagiarism Policy

- All work submitted must be your own. You may not copy from classmates, textbooks, or other resources unless explicitly allowed.
- You may discuss the problem with classmates, but you may not write or debug code with other classmates, other than those in your group.
- If you use any advanced material, you must cite these sources.

Assignment Overview

In this assignment, you will write assembly functions in YASM (Yet Another Assembler) that perform scalar multiplication on a matrix, add two matrices together, and calculate the dot product of two matrices. These tasks are designed to enhance your understanding of matrix manipulation, memory management, and efficient computation at the low-level programming level.

Assignment Objectives

The primary objectives of this assignment are to develop your understanding and skills in matrix manipulation using low-level programming. Specifically, you will:

1. **Perform Scalar Multiplication on a Matrix:**
   - Develop a function that multiplies each element of a matrix by a given scalar value, demonstrating your ability to manipulate matrix elements and understand how matrices are represented in memory.

2. **Add Two Matrices Together:**
    - Create a function that adds corresponding elements of two matrices, which will help you practice memory allocation and handling 2D arrays.
3. **Calculate the Dot Product of Two Matrices:**
    - Implement a function that calculates the dot product of two matrices, reinforcing your understanding of matrix operations and efficient computation.

Task 1: Scalar Multiplication of a Matrix

Your first task is to write a function that multiplies a scalar value to a matrix of any size. The function will take three parameters: a pointer to the matrix, the scalar value, and the dimensions of the matrix (number of rows and columns). The matrix is represented as a 2D array, and you will perform the scalar multiplication using the provided dimensions.

Function Signature

```
void multiplyScalarToMatrix(float **matrix, float scalar, int rows, int cols)
```

Task Description

1. **Multiply the scalar:**
    - Multiply each element in the matrix by the scalar value.
    - Ensure that the resulting matrix is updated in place, meaning the original matrix should be modified with the multiplied values.
2. **Edge Cases:**
    - Handle cases where the matrix has zero elements (i.e., either rows or columns is zero).
    - Ensure that the function does not modify the matrix if the scalar value is 1 (identity operation).

Implementation Details

- Write your assembly code in the file `multiply_scalar_to_matrix.asm`.
- The matrix dimensions (rows and columns) will be provided, so you can directly use these values to iterate through the matrix.

Example

Given a matrix with 2 rows and 2 columns:

$$\begin{pmatrix} 1.0 & 2.0 \\ 3.0 & 4.0 \end{pmatrix}$$

And a scalar value of `2.0`, the function should modify the matrix to:

$$\begin{pmatrix} 2.0 & 4.0 \\ 6.0 & 8.0 \end{pmatrix}$$

Skeleton Code

A partial skeleton has been provided.

Task 2: Adding Two Matrices

Your second task is to write a function that adds two matrices together. The function will take four parameters: pointers to the two matrices, the number of rows, and the number of columns. The function should create a new matrix that contains the result of the addition and return a pointer to this new matrix.

Function Signature

```
float** addMatrices(float **matrix1, float **matrix2, int rows, int cols)
```

Task Description

1. **Input Parameters:**
   - `matrix1`: A pointer to the first matrix (a 2D array of floats).
   - `matrix2`: A pointer to the second matrix (a 2D array of floats).
   - `rows`: The number of rows in both matrices.
   - `cols`: The number of columns in both matrices.
2. **Create a New Matrix:**
   - Allocate memory for a new matrix that will store the result of adding `matrix1` and `matrix2`. The size of this matrix will be `rows x cols`.
3. **Add the Matrices:**
   - Iterate through each element of the matrices, adding the corresponding elements together, and store the result in the newly created matrix.
4. **Return the New Matrix:**
   - Return a pointer to the newly created matrix that contains the result of the addition.
5. **Edge Cases:**
   - Handle cases where the matrices have zero elements (i.e., either `rows` or `cols` is zero).

Implementation Details

- Write your assembly code in the file `add_matrices.asm`.
- You should ensure that memory is properly allocated for the new matrix and that it is deallocated when no longer needed.

Example

Given two matrices with 2 rows and 2 columns:

Matrix 1:

$$\begin{pmatrix} 1.0 & 2.0 \\ 3.0 & 4.0 \end{pmatrix}$$

Matrix 2:

$$\begin{pmatrix} 5.0 & 6.0 \\ 7.0 & 8.0 \end{pmatrix}$$

The function should create a new matrix that contains:

$$\begin{pmatrix} 6.0 & 8.0 \\ 10.0 & 12.0 \end{pmatrix}$$

And return a pointer to this new matrix.

Skeleton Code

A partial skeleton has been provided.

Task 3: Calculating the Dot Product of Two Matrices

Your third task is to write a function that calculates the dot product of two matrices that have the same dimensions. The function will take four parameters: pointers to the two matrices and their dimensions (number of rows and columns). The function should return the resulting dot product as a floating-point number.

Function Signature

```
float calculateMatrixDotProduct(float **matrix1, float **matrix2, int rows, int cols)
```

Task Description

1. **Input Parameters:**
   - `matrix1`: A pointer to the first matrix (a 2D array of floats).
   - `matrix2`: A pointer to the second matrix (a 2D array of floats).
   - `rows`: The number of rows in both matrices.
   - `cols`: The number of columns in both matrices.
2. **Calculate the Dot Product:**
   - Multiply corresponding elements of `matrix1` and `matrix2` and sum the results to compute the dot product.
3. **Return the Dot Product:**
   - Return the computed dot product as a floating-point number.
4. **Edge Cases:**

  - Ensure that both matrices have the same dimensions before performing the calculation.
  - Handle cases where the matrices have zero elements (i.e., either `rows` or `cols` is zero).

Implementation Details

- Write your assembly code in the file `calculate_matrix_dot_product.asm`.
- Ensure that the function is efficient and handles any edge cases appropriately.

Example

Given two matrices with 2 rows and 2 columns:

Matrix 1:

$$\begin{pmatrix} 1.0 & 2.0 \\ 3.0 & 4.0 \end{pmatrix}$$

Matrix 2:

$$\begin{pmatrix} 5.0 & 6.0 \\ 7.0 & 8.0 \end{pmatrix}$$

The function should calculate the dot product as follows:

$$(1.0 \times 5.0) + (2.0 \times 6.0) + (3.0 \times 7.0) + (4.0 \times 8.0) = 70.0$$

The function should return `70.0`.

Skeleton Code

A partial skeleton has been provided.

Mark Distribution

The total marks for this assignment are 15, with the distribution as follows:

- **Task 1: Scalar Multiplication of a Matrix** (4 marks)
  - Correct implementation of scalar multiplication: **2 marks**
  - Handling of edge cases (e.g., zero elements, identity operation): **2 marks**
- **Task 2: Adding Two Matrices** (5 marks)
  - Correct implementation of matrix addition: **3 marks**
  - Proper memory allocation and deallocation: **2 marks**
- **Task 3: Calculating the Dot Product of Two Matrices** (6 marks)
  - Correct calculation of the dot product: **4 marks**
  - Handling of edge cases (e.g., zero elements, matching dimensions): **2 marks**