VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY

UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

REPORT

Assignment

CONVOLUTION OPERATION

Course: Computer Architecture Lab (CO2008)

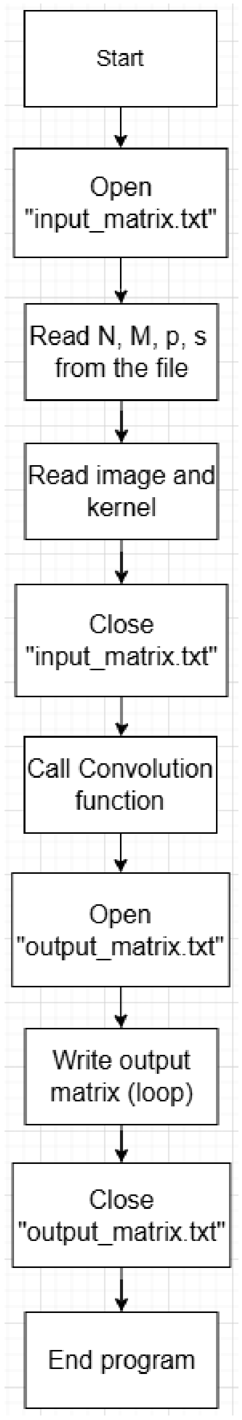
Lecturer: Nguyễn Thành Lộc

Student: Biện Công Thanh – 2053424 (CC11)

HO CHI MINH CITY, NOVEMBER 2024

Table of Contents

1. Flowchart Diagram



1.1. Read N, M, p, s from input file

Read N (Number of rows and columns in the image matrix):

- Read value from the file.
- Convert from float to integer.

Read M (Number of rows and columns in the kernel matrix):

- Read value from the file.
- Convert from float to integer.

Read p (Padding value for image matrix):

- Read value from the file.
- Convert from float to integer.

Read s (Stride value for convolution):

- Read value from the file.
- Convert from float to integer.

1.2. Read Image Matrix

Allocate memory for the image matrix of size *N x N* (since the matrix is square).

Loop through the rows and columns of the image matrix:

- For each element, call ***read_float*** to read the value from the file.
- Store the value in the image matrix.

1.3. Read Kernel Matrix

Allocate memory for the kernel matrix of size *M x M* (since the kernel is square).

Loop through the kernel matrix dimensions:

- For each element in the kernel, ***call read_float*** to read the value from the file.
- Store the value in the kernel matrix.

2. Algorithm for Convolution Calculation

2.1. Function *read_float*

This function reads a float value from the file by reading characters, handling the sign, and parsing the integer and decimal parts. It converts the string to a floating-point value, which is then stored in the $f0 register.

2.2. Function *write_float*

The *write_float* function is designed to convert a floating-point number into its string representation and write it to a file. The function begins by saving all necessary registers onto the stack to preserve their state. It then scales the floating-point number to an integer by multiplying it by 10, rounds it to the nearest whole number, and checks for negative values. The integer part is extracted and converted to a string in reverse order before being written to a buffer, along with a negative sign if applicable. A decimal point is added, followed by the fractional digit. The buffer is then written to the file using a system call. Finally, all saved registers are restored from the stack, ensuring no side effects. This implementation efficiently handles both positive and negative numbers while maintaining precision to one decimal place.

2.3. Function *read_image*

This function reads a matrix (image) from the file. It handles the initialization of the matrix size and clears the image matrix before populating it with values.

It uses a nested loop structure to read and store matrix values row by row and column by column.

2.4. Function *read_kernel*

The *read_kernel* function is responsible for reading a matrix (kernel) from a file and storing its values in memory. The function first initializes the matrix size and clears the kernel matrix before populating it with values. It utilizes a nested loop structure to process the matrix row by row and column by column. Specifically, the outer loop iterates over the rows of the matrix, while the inner loop iterates over the columns. For each element, the function calculates its memory address based on the current row and column indices, using simple arithmetic operations such as multiplication and addition. Once the address is determined, the function calls *read_float* to read the floating-point value from the file and stores it in the corresponding position in the kernel matrix. After completing the loops, the function restores the previously saved register values and returns to the calling function. This approach ensures that the entire matrix is filled with the appropriate values from the input file in a systematic manner.

2.5. Function *convolution*

The *convolution* function is responsible for performing a convolution operation on two matrices: an input image matrix and a kernel matrix, producing an output matrix. It utilizes multiple nested loops to iterate over the elements of the matrices and applies the convolution formula.

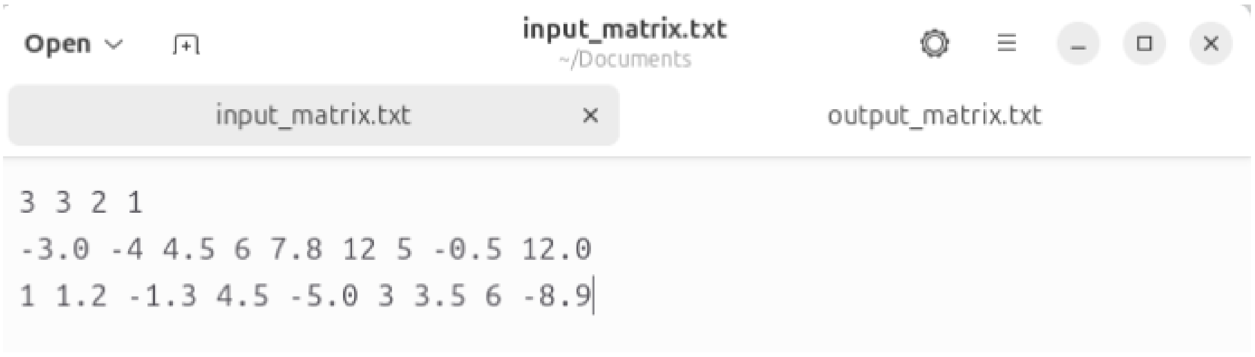
Convolution Loops:

- **Outer Loop** (convolution_loop_1): Iterates over the rows of the output matrix.
- **Second Loop** (convolution_loop_2): Iterates over the columns of the output matrix.
- **Third Loop** (convolution_loop_3): Iterates over the rows of the kernel.
- **Fourth Loop** (convolution_loop_4): Iterates over the columns of the kernel.

3. Result of Sample_test

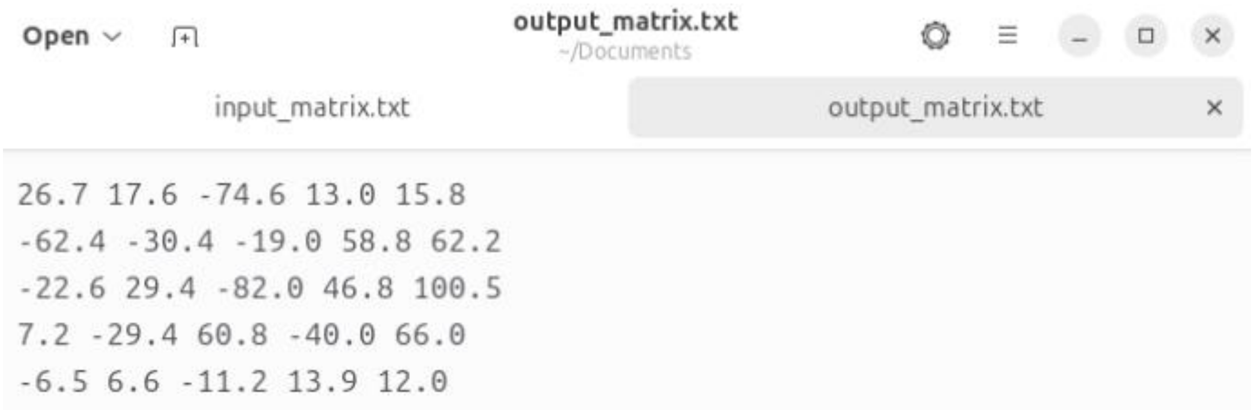
3.1. Test_3

input_matrix.txt:



```
3 3 2 1
-3.0 -4 4.5 6 7.8 12 5 -0.5 12.0
1 1.2 -1.3 4.5 -5.0 3 3.5 6 -8.9
```

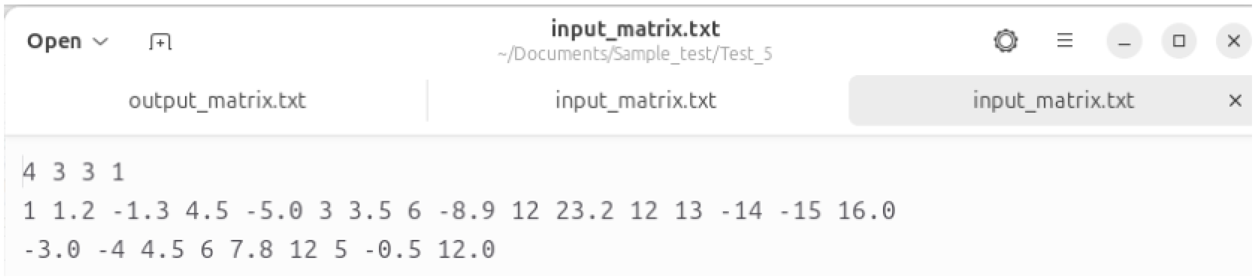
output_matrix.txt:



```
26.7 17.6 -74.6 13.0 15.8
-62.4 -30.4 -19.0 58.8 62.2
-22.6 29.4 -82.0 46.8 100.5
7.2 -29.4 60.8 -40.0 66.0
-6.5 6.6 -11.2 13.9 12.0
```

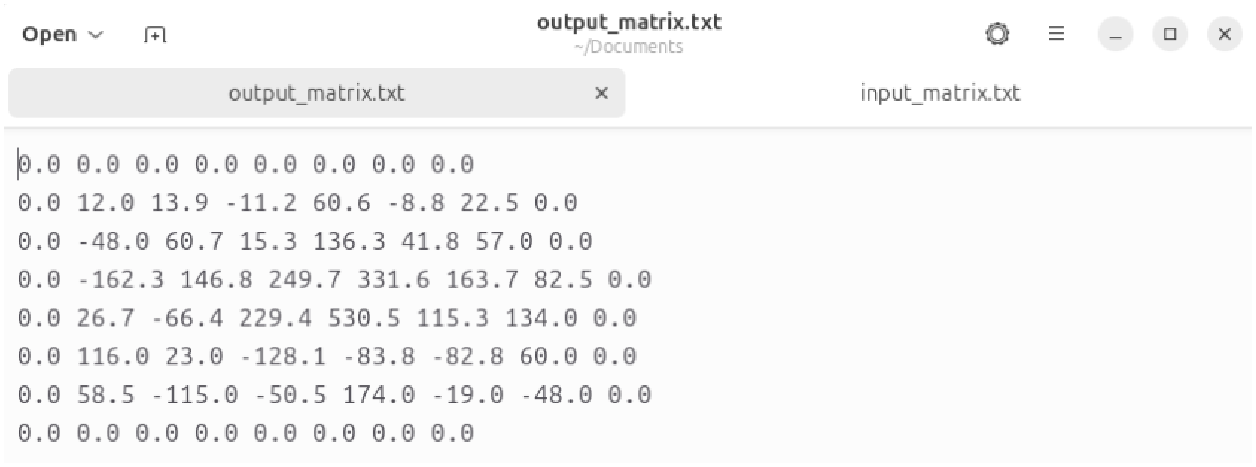
3.2. Test_5

input_matrix.txt:



```
4 3 3 1
1 1.2 -1.3 4.5 -5.0 3 3.5 6 -8.9 12 23.2 12 13 -14 -15 16.0
-3.0 -4 4.5 6 7.8 12 5 -0.5 12.0
```

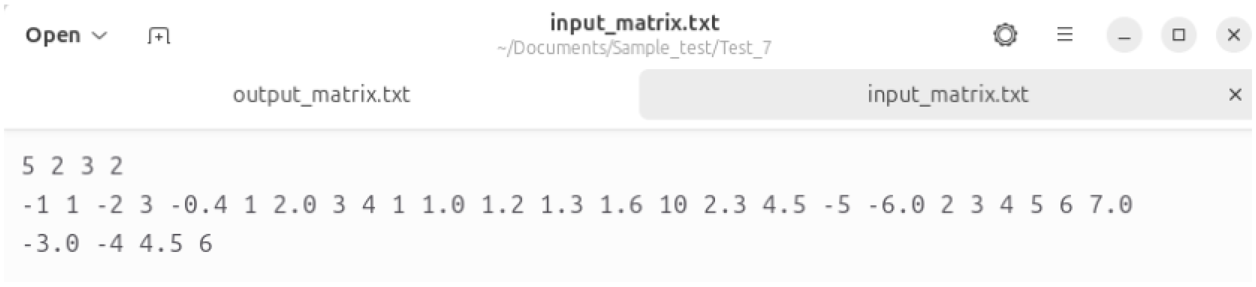
output_matrix.txt:



```
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 12.0 13.9 -11.2 60.6 -8.8 22.5 0.0
0.0 -48.0 60.7 15.3 136.3 41.8 57.0 0.0
0.0 -162.3 146.8 249.7 331.6 163.7 82.5 0.0
0.0 26.7 -66.4 229.4 530.5 115.3 134.0 0.0
0.0 116.0 23.0 -128.1 -83.8 -82.8 60.0 0.0
0.0 58.5 -115.0 -50.5 174.0 -19.0 -48.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

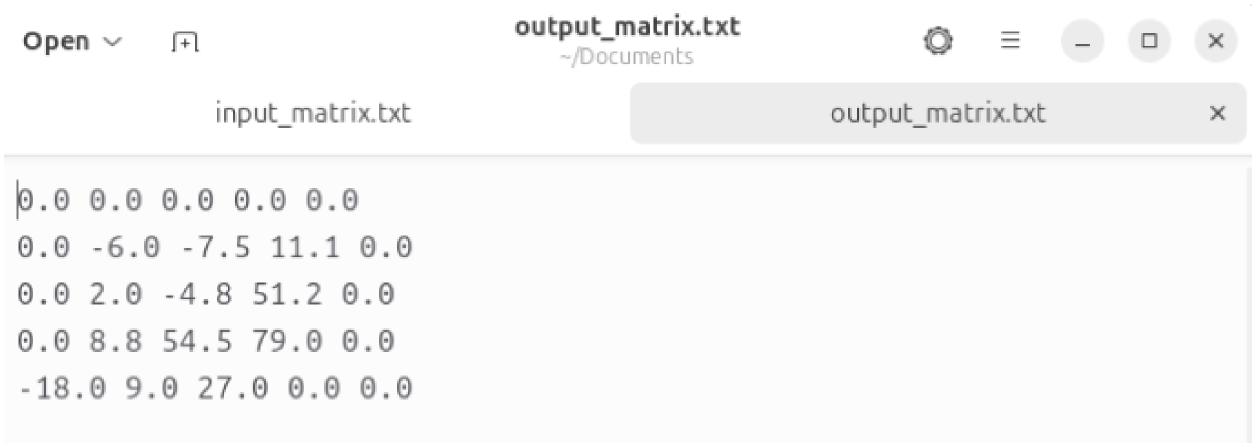
3.3. Test_7

input_matrix.txt:



```
5 2 3 2
-1 1 -2 3 -0.4 1 2.0 3 4 1 1.0 1.2 1.3 1.6 10 2.3 4.5 -5 -6.0 2 3 4 5 6 7.0
-3.0 -4 4.5 6
```

output_matrix.txt:



```
0.0 0.0 0.0 0.0 0.0
0.0 -6.0 -7.5 11.1 0.0
0.0 2.0 -4.8 51.2 0.0
0.0 8.8 54.5 79.0 0.0
-18.0 9.0 27.0 0.0 0.0
```