ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
UNIVERSITY OF WEST ATTICA

DEPARTMENT OF COMPUTER ENGINEERING AND INFORMATION TECHNOLOGY

TASK 2 B.2

CUDA

STUDENT / WORK DETAILS

NAME: ATHANASIOU VASILEIOS EVANGELOS REGISTRATION
NUMBER: 19390005
STUDENT SEMESTER: 11
STUDY PROGRAM: PADA

LABORATORY LEADER: IORDANAKIS MICHALIS
THEORY LEADER: MAMALIS VASILIOS

CONTENTS

PARALLEL SYSTEMS

1. Introduction

1.1 Purpose of the exercise

The purpose of the exercise is to implement a program in CUDA to calculate basic statistics and create the covariance matrix of a two-dimensional array.

1.2 Brief description of the problem being solved

The program manages an N × N array and performs the following operations:

- Calculation of the average of each column.
- Subtracting the average from each element and creating the transfer of the difference table.
- Calculation of the covariance table .

2. Design

2.1 Description of the approach followed

The approach is based on the use of three CUDA cores :

- calcColMeans for calculating averages.
- subMeansT for subtracting means and creating a transfer.
- calcCov to calculate the covariance matrix.

2.2 Analysis of logic and methodologies

Each function is designed to take advantage of parallel execution via CUDA, by dividing the table columns and rows into threads.

2.3 Description of data structures and algorithms

2.3.1 Data structures and variables

- **Input Array (d _ A):** The original N × N array .
- Column **Means (d _ Amean):** NxN element array .
- **Difference Table (d _ Asubmeans): The** N × table resulting after subtracting the mean.
- **Transpose Table (d _ ATsubmeans):** The transpose table of differences.

- **Covariance Table (d _ Acov):** The N × NN \ times table NN × N containing the covariance.

2.3. 2 calcColMeans <<<>>>()

Each thread processes one column of the table to calculate the average.

2.3. 3 subMeansT <<<>>>()

The threads are divided into rows and columns to remove the average and create the transfer. .

2.3. 4 calcCov <<<>>>()

The covariance matrix is calculated for the upper triangular part, reducing the calculations due to symmetry.

3. Implementation

3.1 Reference to the basic functions of the code

Memory allocation and transfer.
Implementation and execution of CUDA cores .
Saving results to files.

3.2 Explanation of parallel parts of the code

The three cores are executed based on the dimGrid and dimBlock configuration , where each thread takes on part of the calculations.

3.3 Description of communication and synchronization between threads

Use shared memory for storing intermediate results.

Synchronization via __syncthreads () .

4. Tests and Results

4.1 Reporting of execution conditions

The execution is done for different array sizes N x N , numbers of threads T per block and blocks per mesh.

The program is compiled via command line in a Linux environment , with the NVIDIA compiler **nvcc** .

```
nvcc - o cuda 2 cuda 2. cu
```

The program is executed via command line in a Linux environment and the user must pass 5 txt files as parameters , so that the table A, the table A _ means with the column averages, the table A _ submeans with the elements of A removed with the column average, the table AT _ submeans the inverse table of A _ submeans and the covariance table A _ cov are stored respectively . Indicative execution command:

```
./ cuda2 A.txt A_means.txt A_submeans.txt AT_submeans.txt A_cov.txt
```

4.2 Presentation of results in text format

The results are stored in the Output folder and tables A and B or C in their respective folders. To save space, not all results are presented in text format in this documentation.

The program requires the user to pass 2 . txt output files with a name of his choice, in which the table A and B or C will be stored . In case the user does not enter the required number of parameters, the program terminates and a characteristic message is displayed

4.2.1 Output_no_args.txt

```
Usage : . /cuda2 A.txt A_means.txt A_submeans.txt AT_submeans.txt A_cov.txt
```

4.2.2 Output8.txt

```
--------------- Device Properties ---------------
Device name: NVIDIA TITAN RTX
Max threads per block: 1024
Max block dimensions: 1024 x 1024 x 64
Max grid dimensions : 2147483647 x 65535 x 65535
-------------------------------------------------
--------------- Input Parameters ---------------
Matrix size: 8 x 8
Blocks per Grid : 2
Threads per Block : 4
------------------------------------------------
The array A has been stored in file A/A8.txt
```

```
The array A_means has been stored in file A_means /A_means8.txt
Time for the kernel calcColMeans <<<>> >( ): 0.253824 ms
The array A_submeans has been stored in file A_submeans /A_submeans8.txt
The array AT_submeans has been stored in file AT_submeans /AT_submeans8.txt
Time for the kernel subMeansT <<<>> >( ): 0.019936 ms
The array A_cov has been stored in file A/A_cov8.txt
Time for the kernel calcCov <<<>> >( ): 0.021216 ms
```

4.2.3 Output512.txt

```
--------------- Device Properties ---------------
Device name: NVIDIA TITAN RTX
Max threads per block: 1024
Max block dimensions: 1024 x 1024 x 64
Max grid dimensions : 2147483647 x 65535 x 65535
-------------------------------------------------
--------------- Input Parameters ---------------
Matrix size: 512 x 512
Blocks per Grid: 32
Threads per Block: 16
-----------------------------------------------
The array A has been stored in file A/A512.txt
The array A_means has been stored in file A_means /A_means512.txt
Time for the kernel calcColMeans <<<>> >( ): 0.124000 ms
The array A_submeans has been stored in file A_submeans /A_submeans512.txt
The array AT_submeans has been stored in file AT_submeans /AT_submeans512.txt
Time for the kernel subMeansT <<<>> >( ): 0.018880 ms
The array A_cov has been stored in file A/A_cov512.txt
Time for the kernel calcCov <<<>> >( ): 0.885408 ms
```

4.2.4 Output1024.txt

```
--------------- Device Properties ---------------
Device name: NVIDIA TITAN RTX
Max threads per block: 1024
Max block dimensions: 1024 x 1024 x 64
Max grid dimensions : 2147483647 x 65535 x 65535
-------------------------------------------------
--------------- Input Parameters ---------------
Matrix size: 1024 x 1024
Blocks per Grid: 32
Threads per Block : 32
-----------------------------------------------
The array A has been stored in file A/A1024.txt
The array A_means has been stored in file A_means /A_means1024.txt
Time for the kernel calcColMeans <<<>> >( ): 0.159168 ms
The array A_submeans has been stored in file A_submeans /A_submeans1024.txt
The array AT_submeans has been stored in file AT_submeans /AT_submeans1024.txt
Time for the kernel subMeansT <<<>> >( ): 0.071616 ms
```

```
The array A_cov has been stored in file A/A_cov1024.txt
Time for the kernel calcCov <<<>> >( ): 11.949280 ms
```

4.2.5 Output10000.txt

```
--------------- Device Properties ---------------
Device name: NVIDIA TITAN RTX
Max threads per block: 1024
Max block dimensions: 1024 x 1024 x 64
Max grid dimensions : 2147483647 x 65535 x 65535
--------------------------------------------------
--------------- Input Parameters ---------------
Matrix size: 10000 x 10000
Blocks per Grid: 100
Threads per Block: 100
------------------------------------------------
The array A has been stored in file A/A10000.txt
The array A_means has been stored in file A_means /A_means10000.txt
Time for the kernel calcColMeans <<<>> >( ): 1.065632 ms
The array A_submeans has been stored in file A_submeans /A_submeans10000.txt
The array AT_submeans has been stored in file AT_submeans
/AT_submeans10000.txt
Time for the kernel subMeansT <<<>> >( ): 0.009952 ms
The array A_cov has been stored in file A/A_cov10000.txt
Time for the kernel calcCov <<<>> >( ): 0.124096 ms
```

4.3 Efficiency analysis

4.3.1 Execution times of the parallel algorithm

Below are the execution times of the three basic kernels of the algorithm , calcColMeans **,** subMeansT **,** and **calcCov** , for different dimensions of the AAA matrix .

Dimension Table AAA	**calcColMeans (ms)**	**subMeansT (ms)**	**calcCov (ms)**
8x8	0.253824	0.019936	0.021216
512x512	0.124000	0.018880	0.885408
1024x1024	0.159168	0.071616	11.949280
10000 x 10000	1.065632	0.009952	0.124096

4.3.3 Observations

- **Increasing the Table Dimension:**

    - The execution time of the **calcColMeans kernel** increases linearly with the dimension of the array, as expected, since it processes each column independently.
    - The execution time of **subMeansT remains constant or decreases slightly, likely due to better utilization of** GPU resources on larger arrays.
    - **calcCov** kernel exhibits a rapid increase in execution time when the matrix dimension increases, especially for sizes 1024x1024 due to the increased complexity of computing the covariance matrix.

- **Effect of Grid and Block Structure:**

    - Proper grid and block design significantly impacts performance, particularly for larger array sizes.
    - For smaller array sizes, the layout effect is smaller, as the number of threads is limited.

- **If efficiency for adults Tables :**

    - For the 10000×1000010000 matrix, the **calcCov kernel** completes faster, while inconsistencies in execution time are observed. This may be due to different resource usage or grid and block adaptation. size for large paintings.

- **Stability and Materialization :**

    - **subMeansT** time with increasing dimension is due to the use of shared memory and the effective use of synchronization.
    - Times for small arrays (e.g. 8×8) are significantly longer relative to the data size, due to the general delay introduced by kernel startup .

5. Problems and Solutions

5.1 Reporting problems

Synchronization Problem:

During the implementation of the code, a synchronization problem was identified between threads during the execution of kernels . More specifically :

1. **Inconsistent Access to Shared Memory:**
   - When using **shared memory** , some threads were attempting to read or write data simultaneously, resulting in inconsistent values in intermediate operations.
   - In cases with large grids or blocks, the lack of proper synchronization within the blocks led to incorrect calculations.
2. **Lack of Synchronization After Parallel Reduction :**
   - In methods like **calcAvg** and **findMax** , the problem appeared in the final phase of the reduction . Threads that had not completed their work would synchronize their values later than the rest, resulting in the final result being incorrect.
3. **Writing and Reading Asynchronous Tables:**
   - When calculating the covariance matrix (**calcCov**), threads were attempting to read values from other columns before previous threads had completed their operations.

Thank you for your attention.

PARALLEL SYSTEMS