PARALLEL SYSTEMS

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ
UNIVERSITY OF WEST ATTICA

DEPARTMENT OF COMPUTER ENGINEERING AND INFORMATION TECHNOLOGY

TASK 2 B.1

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

PARALLEL SYSTEMS

CONTENTS

PARALLEL SYSTEMS

1. Introduction

1.1 Purpose of the exercise

The purpose of the exercise is to use the CUDA architecture to solve a matrix calculation problem, utilizing parallel processing.

1.2 Brief description of the problem being solved

The program creates a random 2D array of integers and performs the following operations:

- Calculation medium term (calcAvg).
- Finding maximum element (findMax).
- Create array B based on the elements of the original array (createB).
- Create array C based on adjacent elements (createC).

2. Design

2.1 Description of the approach followed

The approach involves parallelizing basic operations on arrays using CUDA kernels. Computations are distributed across threads via grids and blocks.

2.2 Analysis of logic and methodologies

The logic is based on the following:

• Use of parallel reductions to calculate average and find maximum.
• Use shared memory to improve performance.
• Dividing the table into sections to distribute the work to threads .

2.3 Description of data structures and algorithms

2.3.1 Data structures and variables

Basic data structures include:

- Input (d _ A) and output (d _ OutArr) arrays in device memory (device).
- Variables for storing average, maximum and minimum items.
- Structures for creating CUDA events for time measurement.

2.3.2 Generator program for producing 2D arrays

create2DArray function creates a random array of size NxN, ensuring that its maximum element is greater than N · m, where m is the average.

2.3.3 calcAvg <<<>>>()

The **kernel function** calculates the average of all elements of the array using reduction and atomic commands.

2.3.4 findMax <<<>>>()

The **kernel function** finds the maximum element of the array with parallel reduction and the use of atomic instructions.

2.3.5 createB <<<>>>()

It calculates the matrix Bij=(m−Aij) / amax and finds the minimum element of the matrix.

2.3.6 createC <<<>>>()

It creates the matrix Cij =(Aij + Ai (j +1)+ Ai (j −1)) / 3 , taking into account the boundaries of the matrix.

2.3.7 atomicMin

atomicMin is a custom function used to calculate the minimum value in floating point numbers on the device. Its use is necessary, as CUDA does not provide support for atomic instructions on floating point numbers (floats). It is implemented with atomicCAS

3. Implementation

3.1 Reference to the basic functions of the code

The code of conduct includes :

- Initializing CUDA data and parameters .
- Creation and transfer of data from host to device .
- Execution of the above CUDA kernels .

- Retrieval and storage of results .

3.2 Explanation of parallel parts of the code

The use of shared memory improves the speed of calculations by reducing access to global memory.
Parallel operations distribute the work into threads and blocks .

3.3 Description of communication and synchronization between threads

Use __syncthreads ( ) to synchronize threads in the same block.

Use atomic operations to safely modify shared data.

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 1 cuda 1. cu
```

The program is executed via command line in a Linux environment and the user must pass 2 txt files as parameters , so that table A and table B or C are saved respectively . Indicative execution command:

```
./ cuda 1 A . txt OutArr.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 : . /cuda1 A.txt OutArr.txt
```

To check the correctness of the algorithm, we first tested a small-sized table to confirm that classification is achieved.

4.2.2 Output8B.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/A8B.txt
Average: 64.41
Time for the kernel calcAvg <<<>> >( ): 0.204736 ms
Max: 624
Time for the kernel findMax <<<>> >( ): 0.015552 ms
The array B has been stored in file OutArr /OutArr8B.txt
Min: -0.0522
Time for the kernel createB <<<>> >( ): 0.015040 ms
```

4.2.3 Output8C.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/A8C.txt
Average: 66.98
Time for the kernel calcAvg <<<>> >( ): 0.207712 ms
Max: 444
Time for the kernel findMax <<<>> >( ): 0.014592 ms
The array C has been stored in file OutArr /OutArr8C.txt
Time for the kernel createC <<<>> >( ): 0.012992 ms
```

4.2.4 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
Average: 3.12
Time for the kernel calcAvg <<<>> >( ): 0.136576 ms
Max: 100
Time for the kernel findMax <<<>> >( ): 0.016704 ms
The array C has been stored in file OutArr /OutArr512.txt
Time for the kernel createC <<<>> >( ): 0.016576 ms
```

4.2.5 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
Average: 1.52
Time for the kernel calcAvg <<<>> >( ): 36.310913 ms
Max: 51241
```

```
Time for the kernel findMax <<<>> >( ): 0.059424 ms
The array B has been stored in file OutArr /OutArr1024.txt
Min: -0.0019
Time for the kernel createB <<<>> >( ): 0.072832 ms
```

4.2.6 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
Average: 0.00
Time for the kernel calcAvg <<<>> >( ): 29.475456 ms
Max: 0
Time for the kernel findMax <<<>> >( ): 0.014784 ms
The array C has been stored in file OutArr /OutArr10000.txt
Time for the kernel createC <<<>> >( ): 0.011424 ms
```

4.2.7 Output20000.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: 20000 x 20000
Blocks per Grid: 162
Threads per Block : 124
------------------------------------------------
The array A has been stored in file A/A20000.txt
Average: 0.00
Time for the kernel calcAvg <<<>> >( ): 39.388447 ms
Max: 0
Time for the kernel findMax <<<>> >( ): 0.015104 ms
The array C has been stored in file OutArr /OutArr20000.txt
Time for the kernel createC <<<>> >( ): 0.011424 ms
```

4.3 Efficiency analysis

4.3.1 Execution times of the parallel algorithm

The execution times increase linearly with the array size for calcAvg , as expected due to the calculation traversing all elements. The performance for findMax and createC remains constant due to the use of optimized parallel methods.

Size Pin aka	calcAvg (ms)	findMax (ms)	createB / createC (ms)
8x8	0.204736	0.015552	0.015040 (B)
512x512	0.136576	0.016704	0.016576 (C)
1024x1024	36.310913	0.059424	0.072832 (B)
10000 x 10000	29.475456	0.014784	0.011424 (C)
20000x20000	39.388447	0.015104	0.011424 (C)

4.3.3 Observations

The execution times for calcAvg show an increasing trend for larger array sizes, as expected.

The use of atomic instructions in the kernels (findMax , createB) ensures constant times regardless of the size of the array.
The performance of the createC kernel remains extremely high even for large arrays, thanks to parallel processing.

5. Problems and Solutions

5.1 Reporting problems

1. **Memory Allocation Problem for Large NNNs :**
   When running for large arrays (e.g., 512×512), a failure occurred while transferring data from device memory to host memory :
    - **Error :** CUDA Error --> cudaMemcpy (h_avg , d_avg , sizeof (float), cudaMemcpyDeviceToHost) failed. cudamemcpy_error
    - **Cause:** Insufficient memory to store large tables on the device or poor memory management between host and device .
2. **Problem Using Shared Memory :**
   When compiling the code, the calcAvg , findMax , and createB kernels failed due to exceeding the maximum amount of shared memory available per block:
    - **Error :**

      vbnet
      Copy code

```
ptxas error: Entry function '_Z7createBPiPfS0_S_S0_' uses too much shared data (0x400000 bytes, 0x18000 max)
```

shared_mem_err

- **Reason: The definition of large** shared tables memory for each block exceeds the maximum allowed capacity of the device.

3. **Thread Synchronization Issues:**
   When executing parallel operations using shared memory , difficulties were observed in properly synchronizing threads:
   - __syncthreads () was not used sufficiently or correctly, causing inconsistent results in some cases.

5.2 Solutions tested and implemented

For the Memory Allocation Problem:

- **Data - Related Optimization :**
  - Data transfers from the host to the device and vice versa were broken into smaller chunks, reducing the chance of exceeding available memory.
- **Control Size Tables :**
  - Dynamic control was implemented to ensure that N did not exceed the device limits.

- **For a Synchronization Issues Yarn :**

  - **Add `__syncthreads ()` :**
    - Ensured that all threads in the block have completed loading data into the shared memory before starting calculations.

Thank you for your attention.