

CS6023: GPU Programming

Assignment 2

Released: 23 August 2026 | Due: 13 September 2026, 23:59 IST | Marks: 12

1 Problem Statement

Given four integer matrices $A_{q \times p}$, $B_{q \times r}$, $C_{p \times q}$ and $D_{r \times q}$, efficiently compute the matrix E

$$E = A^T B + C D^T \quad (1)$$

by considering the aspects of **memory coalescing** and **shared memory**.

Note that A is stored with q rows and p columns, so A^T is $p \times q$; similarly D is stored with r rows and q columns, so D^T is $q \times r$. The result E is $p \times r$.

2 Input and Output

2.1 Input format

- The first line of the input contains 3 integers p , q and r .
- The next q lines contain p space-separated integers, representing matrix A .
- The next q lines contain r space-separated integers, representing matrix B .
- The next p lines contain q space-separated integers, representing matrix C .
- The next r lines contain q space-separated integers, representing matrix D .

2.2 Output format

- Write p lines, each containing r space-separated integers, representing the matrix $E_{p \times r}$.

2.3 Constraints

- $2 \leq p, q, r \leq 1024$.
- Each integer of the matrices A , B , C and D is in the range $[-10, 10]$.

3 Sample Testcase

Let $p = 2$, $q = 3$, $r = 2$.

$$A = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \quad B = \begin{bmatrix} 1 & -2 \\ -4 & 3 \\ 5 & -6 \end{bmatrix} \quad C = \begin{bmatrix} 4 & 2 & 10 \\ 7 & 5 & -8 \end{bmatrix} \quad D = \begin{bmatrix} 8 & 1 & 5 \\ -3 & 3 & -6 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad D^T = \begin{bmatrix} 8 & -3 \\ 1 & 3 \\ 5 & -6 \end{bmatrix}$$

$$A^T B = \begin{bmatrix} 8 & -14 \\ 14 & -29 \end{bmatrix} \quad C D^T = \begin{bmatrix} 84 & -66 \\ 21 & 42 \end{bmatrix}$$

$$\Rightarrow E = A^T B + C D^T = \begin{bmatrix} 92 & -80 \\ 35 & 13 \end{bmatrix}$$

4 Points to be Noted

- The file `main.cu` provided by us contains the code, which takes care of taking the input, printing the result, and printing the execution time.
- Don't write any code in the `main()` function. Do not write any print statements.
- You need to implement the `compute()` function provided in the `main.cu`.
- You are free to use any number of functions/kernels.
- You can launch the kernels as you wish.
- Test your code on large input matrices.
- **It is compulsory to optimize for coalesced accesses. Also, make use of shared memory.**

5 What You Are Given

The assignment folder on Moodle has the following contents. Keep this layout intact; the scripts locate everything relative to the assignment folder.

```
Assignment2/
|-- Assignment2.pdf          this document
|-- README.md               step-by-step instructions
|-- main.cu                 starter code -- implement compute() here
|-- submit/                 put your <RollNumber>.cu in here
|-- scripts/
|   |-- run_tests.sh        runs the 6 public test cases
|   |-- aqua_job.cmd         job script for the Aqua cluster
|   |-- compile.sh          builds one file (for debugging)
|-- testcases/
|   |-- input/              input1.txt ... input6.txt
|   |-- output/             output1.txt ... output6.txt    (expected results)
```

The six test cases in `testcases/` are the *public* ones, so you can check your own work. Your final marks are computed on a larger set of *private* test cases that follow the same input and output format and obey the same constraints.

6 How to Run and Test Your Code

Step 1 (common to both options). Copy the starter file into `submit/`, renaming it to your roll number, and implement `compute()` inside that copy. For example, if your roll number is `cs22d003`:

```
cd Assignment2
cp main.cu submit/cs22d003.cu
chmod +x scripts/*.sh
```

Then pick **one** of the two options below, depending on the machine you have.

6.1 Option A - a machine with an NVIDIA GPU

Requirements: an NVIDIA GPU, a working NVIDIA driver, and the CUDA Toolkit (so that `nvcc` is on your `PATH`).

```
./scripts/run_tests.sh          # compiles submit/*.cu and runs all 6 test cases
```

Useful variations while debugging:

```
./scripts/run_tests.sh -c 3      # run only test case 3
./scripts/run_tests.sh -v      # on failure, show expected vs. your output
./scripts/run_tests.sh -h      # show all options
```

6.2 Option B - the Aqua cluster

Use this if you do not have an NVIDIA GPU. Copy the assignment folder to Aqua and submit a job. Run the first command *on your own machine*, from the directory that contains the Assignment2 folder:

```
scp -P 40826 -r Assignment2 <your-roll-no>@aqua.iitm.ac.in:~/
ssh <your-roll-no>@aqua.iitm.ac.in -p 40826
```

Aqua does not listen on the default SSH port, so the port option is required in both commands. Note the case: `scp` takes a capital `-P` before the file names, while `ssh` takes a small `-p`. Aqua is also reachable only from inside the institute network, so connect to the IITM VPN first if you are working from outside the campus.

After `ssh` you land in your home directory on Aqua, which is where the folder was just copied. Submit the job from inside it:

```
cd Assignment2
chmod +x scripts/*.sh
qsub scripts/aqua_job.cmd
```

`qsub` prints a job id and returns immediately - the job is queued, not finished. Check on it, and read the results once it is done:

```
qstat -u <your-roll-no>      # empty output means the job has finished
cat results.txt              # the PASS/FAIL table
cat logfile.log              # everything the job printed
cat errorfile.err            # error messages, if any
```

Important: do not run `nvcc` or `run_tests.sh` directly on the Aqua login node - it has no GPU, and the login node is shared by everybody. Always go through `qsub`.

6.3 Reading the output

Both options run the same script and print the same table. PASS means your output matched the expected output exactly; FAIL means it did not.

```
=== submit/cs22d003.cu ===
Build OK
```

TEST	RESULT	TIME	DETAIL
input1	PASS	0.412 ms	output matches
input2	PASS	0.395 ms	output matches
...			
cs22d003: 6/6 test cases passed			

7 Submission Guidelines

- Submit **one zip file** on Moodle, named `<YourRollNumber>.zip`, containing **exactly one** file: your `<YourRollNumber>.cu`, with no folder around it. Do not include the `submit/` folder,

the test cases, or any other file:

```
cd submit
zip CS22D003.zip CS22D003.cu      # produces CS22D003.zip -> CS22D003.cu
```

- The roll number may be in **upper or lower case**; both are accepted. Use the same spelling for the zip and the .cu inside it.
- After submitting, download the zip again, open it, and confirm that it holds the file you intended to submit.
- Your file must compile with `nvcc` and pass the test cases as it is, with no manual edits by the evaluator.

8 Learning Objectives

- Write a CPU version of the code achieving the same functionality. Time the CPU code and the GPU code separately for large matrices and compare the performances.
- Try to use a minimum number of kernel calls.
- Try to use `__syncthreads()` and `__syncwarp()` as minimally as possible to gain performance benefits.
- Exploit shared memory as much as possible to gain performance benefits.
- Think of an approach or technique that can avoid bank conflicts of shared memory.
- To understand how good coalescing and shared-memory utilisation are, use the NVIDIA profilers to see the effective memory bandwidth on large test cases.