**Class:** Final Year (Computer Science and Engineering)

**Year:** 2021-22 **Semester:** 1 **Date:** 21/11/2021

**Course:** High Performance Computing Lab

**Practical No. 8**

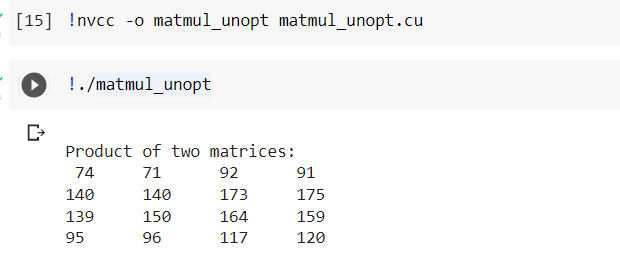
**Exam Seat No:**

2018BTECS00071 – Saurabh Makarand Narkhede

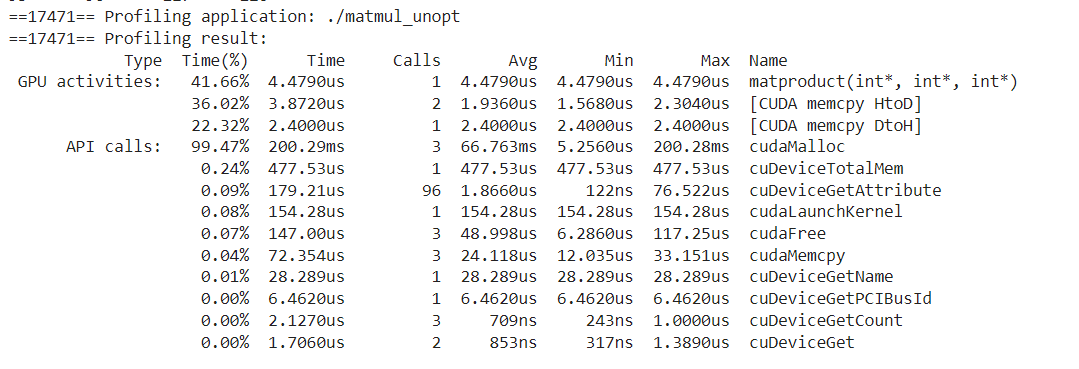
**GitHub Link:**

**Problem Statement 1:**

Write a CUDA C program to perform the simple matrix-matrix multiplication. Perform code optimization and profiling of existing CUDA C program.

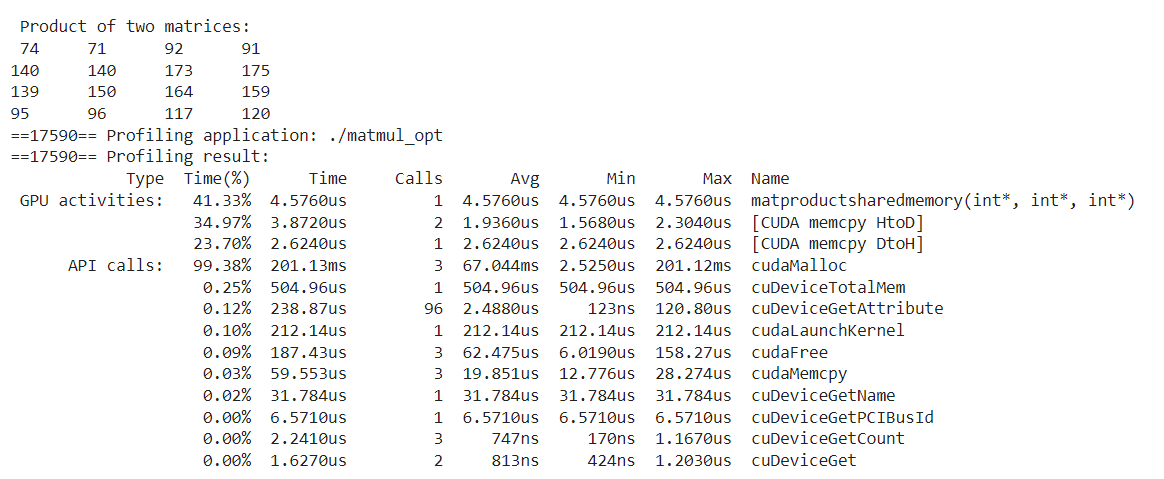


**Unoptimized CUDA program Profiling**

****

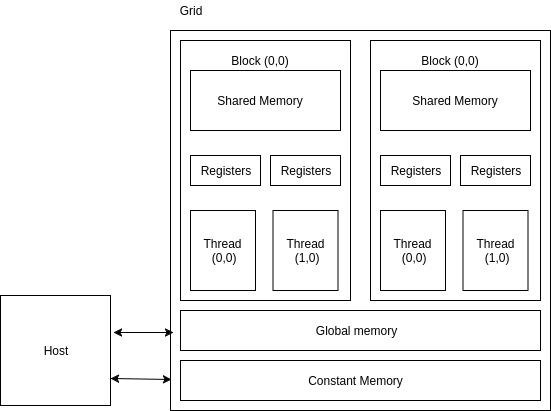
 I have considered two matrices of sizes row1\*col1 and row2\*col2. Resultant matrix(product), definitely will be of size row1\*col2. That's why, I have considered a two dimensional grid having row1\*col2 blocks. Each block will be responsible to calculate one value of product. To find each value in product, there will be col1 or row2 number of multiplications. So I considered, Col1 number of threads in each block; So one thread for one multiplication. In short, total number of blocks = row1\*col2 and in each block, number of threads = col1(or row2).  
           I have used shared array p[] to save intermediate multiplication values. In each block, each thread will find out value of p[i]. \_\_syncthreads() will make sure that all threads has finished their computation i.e. all values of array p[] are available which will be added together to get one value of product.

**Optimized(using shared memory) CUDA program Profiling**

****

**Problem Statement 2:**

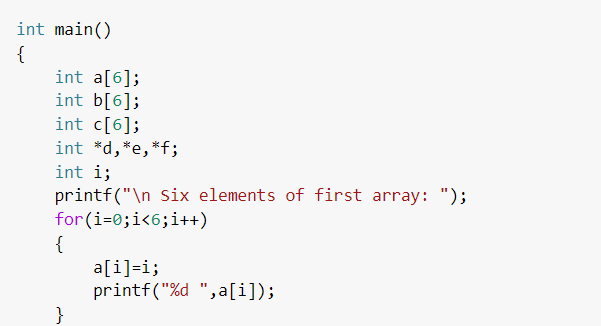
Write a CUDA C program to demonstrate the use of different GPU memories.

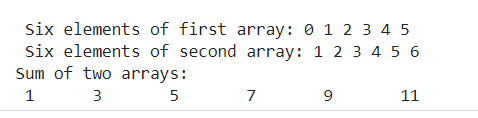


In terms of speed, if all the various types of device memory were to race here’s how the race would turn out:

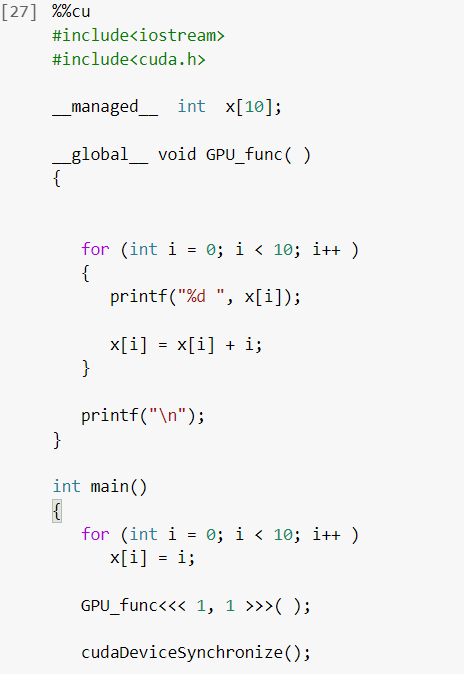
* 1st place: Register file
* 2nd place: Shared Memory
* 3rd place: Constant Memory
* 4th: Texture Memory
* Tie for last place: Local Memory and Global Memory
* Data stored in **register memory** is visible only to the thread that wrote it and lasts only for the lifetime of that thread.
* **Local memory** has the same scope rules as register memory, but performs slower.
* Data stored in **shared memory** is visible to all threads within that block and lasts for the duration of the block. This is invaluable because this type of memory allows for threads to communicate and share data between one another.
* Data stored in **global memory** is visible to all threads within the application (including the host), and lasts for the duration of the host allocation.
* *Constant and texture memory won’t be used here because they are beneficial for only very specific types of applications.* **Constant memory** is used for data that will not change over the course of a kernel execution and is read only. Using constant rather than global memory can reduce the required memory bandwidth, however, this performance gain can only be realized when a warp of threads read the same location.Similar to constant memory, **texture memory** is another variety of read-only memory on the device. When all reads in a warp are physically adjacent, using texture memory can reduce memory traffic and increase performance compared to global memory.

**• Use of private memory.**

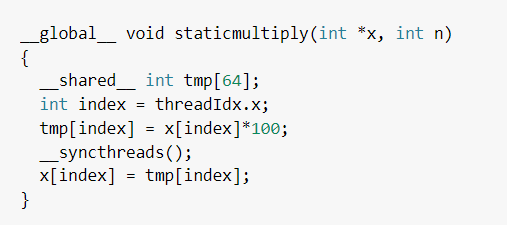
****

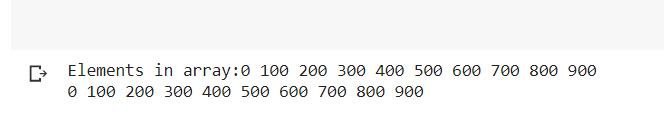
****

**• Use of shared memory.**

****

**• Use of global memory.**

****

****

All in all, for most applications my recommendation is definitely to try to make use of shared memory wherever possible. It is the most versatile and easy-to-use type of memory. Shared memory allows communication between threads within a warp which can make optimizing code much easier for beginner to intermediate programmers. The other types of memory all have their place in CUDA applications, but for the general case, shared memory is the way to go.