

Walchand College of Engineering, Sangli  
Department of Computer Science and Engineering  
**Class:** Final Year (Computer Science and Engineering)  
**Year:** 2021-22                      **Semester:** 1  
**Course:** High Performance Computing Lab

## Practical No. 8

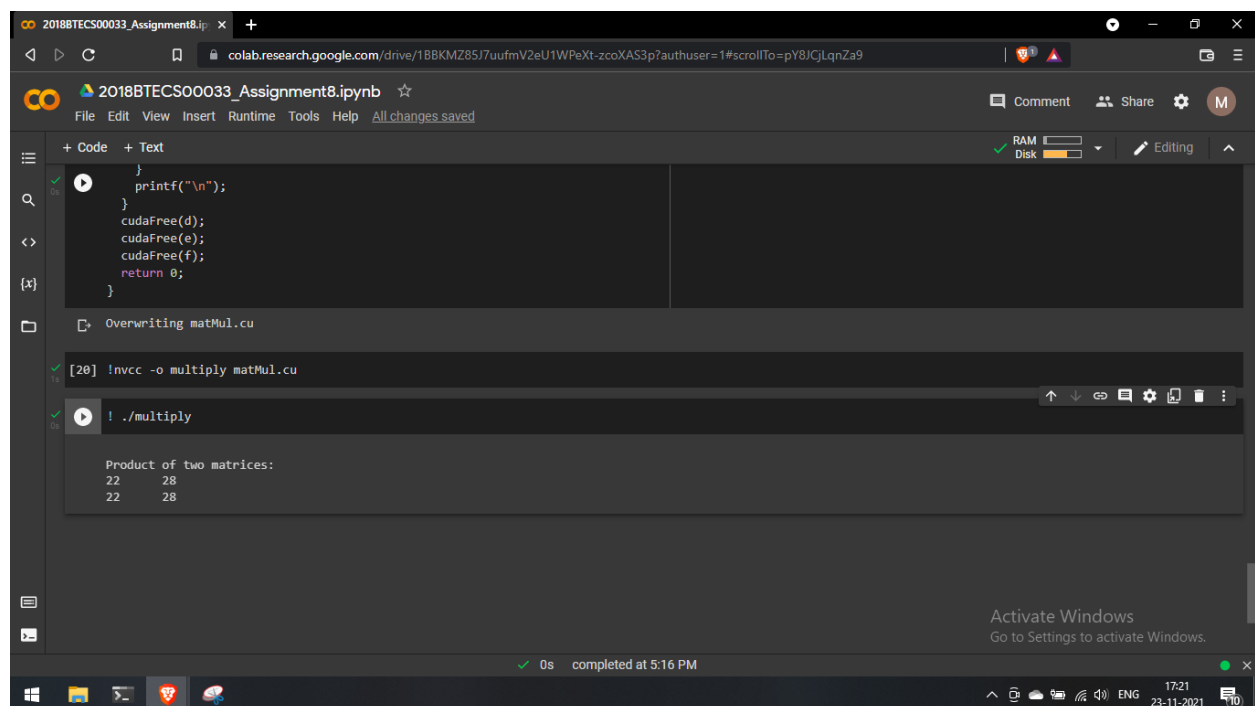
### Exam Seat No:

1. 2018BTECS00033 - Mahendra Bhimrao Garge

### 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. (Attach Snapshot of execution before optimization and after optimization)**

### Screenshot 1:



```
2018BTECS00033_Assignment8.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
}
printf("\n");
}
cudaFree(d);
cudaFree(e);
cudaFree(f);
return 0;
}

Overwriting matMul.cu

[20] !nvcc -o multiply matMul.cu

!./multiply

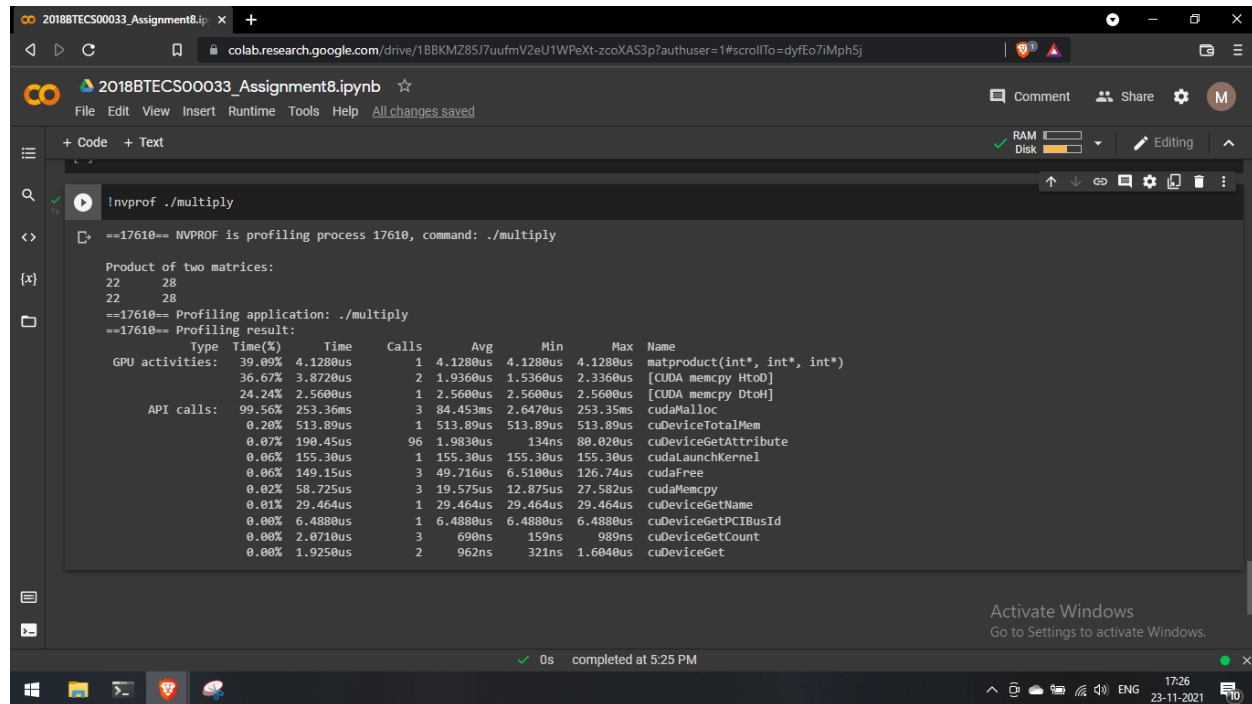
Product of two matrices:
22 28
22 28

Activate Windows
Go to Settings to activate Windows.

0s completed at 5:16 PM
```

### Information 1: Simple matrix multiplication

## Screenshot 2:



```
2018BTCS00033_Assignment8.ipynb
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text
RAM Disk
Editing

In[ ]: nvprof ./multiply

==17610== NVPROF is profiling process 17610, command: ./multiply

Product of two matrices:
22 28
22 28

==17610== Profiling application: ./multiply
==17610== Profiling result:

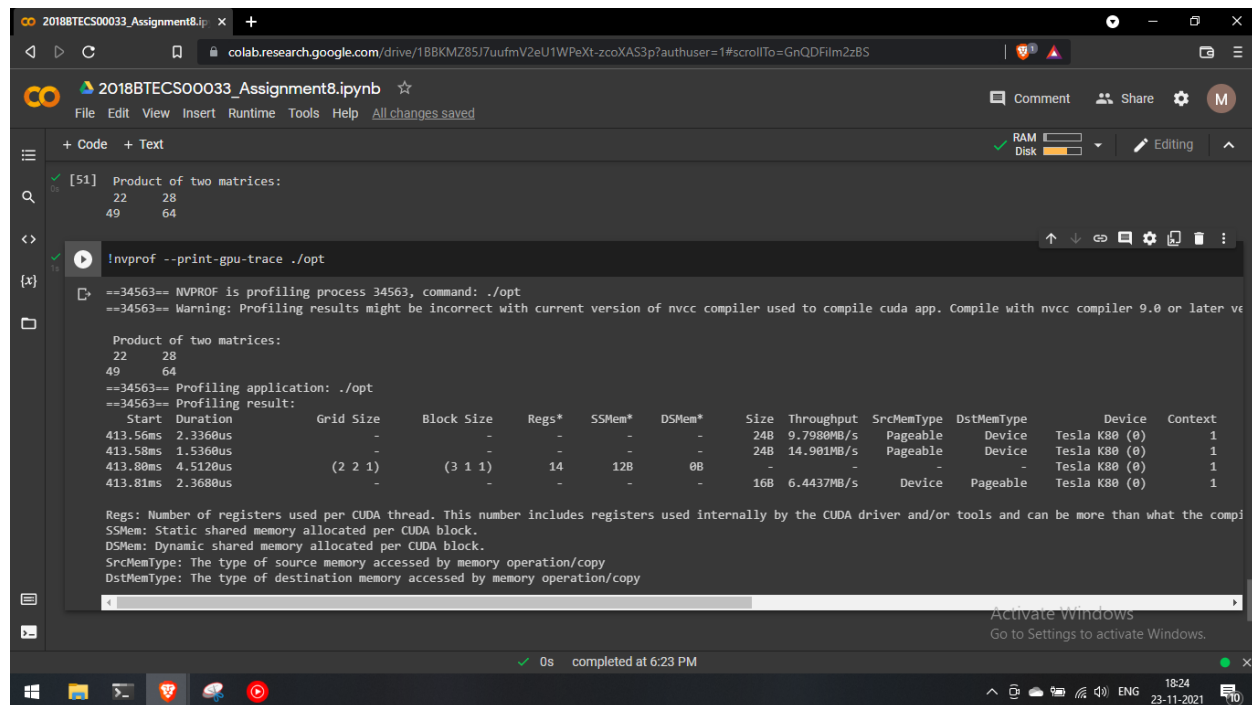
Type Time(%) Time Calls Avg Min Max Name
GPU activities: 39.09% 4.1280us 1 4.1280us 4.1280us 4.1280us matproduct(int*, int*, int*)
36.67% 3.8720us 2 1.9360us 1.5360us 2.3360us [CUDA memcpy HtoD]
24.24% 2.5600us 1 2.5600us 2.5600us 2.5600us [CUDA memcpy DtoH]
API calls: 99.56% 253.36ms 3 84.453ms 2.6470us 253.35ms cudaMalloc
0.20% 513.89us 1 513.89us 513.89us 513.89us cuDeviceTotalMem
0.07% 190.45us 96 1.9830us 134ns 80.020us cuDeviceGetAttribute
0.06% 155.30us 1 155.30us 155.30us 155.30us cudaLaunchKernel
0.06% 149.15us 3 49.716us 6.5100us 126.74us cudaFree
0.02% 58.725us 3 19.575us 12.875us 27.582us cudaMemcpy
0.01% 29.464us 1 29.464us 29.464us 29.464us cuDeviceGetName
0.00% 6.4880us 1 6.4880us 6.4880us 6.4880us cuDeviceGetPCIBusId
0.00% 2.0710us 3 690ns 159ns 989ns cuDeviceGetCount
0.00% 1.9250us 2 962ns 321ns 1.6040us cuDeviceGet

Activate Windows
Go to Settings to activate Windows.

0s completed at 5:25 PM
```

## Information 2: Screenshot of profiling before optimization

## Screenshot 3:



```
2018BTCS00033_Assignment8.ipynb
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text
RAM Disk
Editing

In[51]: Product of two matrices:
22 28
49 64

In[ ]: nvprof --print-gpu-trace ./opt

==34563== NVPROF is profiling process 34563, command: ./opt
==34563== Warning: Profiling results might be incorrect with current version of nvcc compiler used to compile cuda app. Compile with nvcc compiler 9.0 or later version.

Product of two matrices:
22 28
49 64

==34563== Profiling application: ./opt
==34563== Profiling result:

Start Duration Grid Size Block Size Regs* SSMem* DSMem* Size Throughput SrcMemType DstMemType Device Context
413.56ms 2.3360us - - - - - 24B 9.7980MB/s Pageable Device Tesla K80 (0) 1
413.58ms 1.5360us - - - - - 24B 14.901MB/s Pageable Device Tesla K80 (0) 1
413.80ms 4.5120us (2 2 1) (3 1 1) 14 128 0B - - Tesla K80 (0) 1
413.81ms 2.3680us - - - - - 16B 6.4437MB/s Device Pageable Tesla K80 (0) 1

Regs: Number of registers used per CUDA thread. This number includes registers used internally by the CUDA driver and/or tools and can be more than what the compiler reports.
SSMem: Static shared memory allocated per CUDA block.
DSMem: Dynamic shared memory allocated per CUDA block.
SrcMemType: The type of source memory accessed by memory operation/copy
DstMemType: The type of destination memory accessed by memory operation/copy

Activate Windows
Go to Settings to activate Windows.

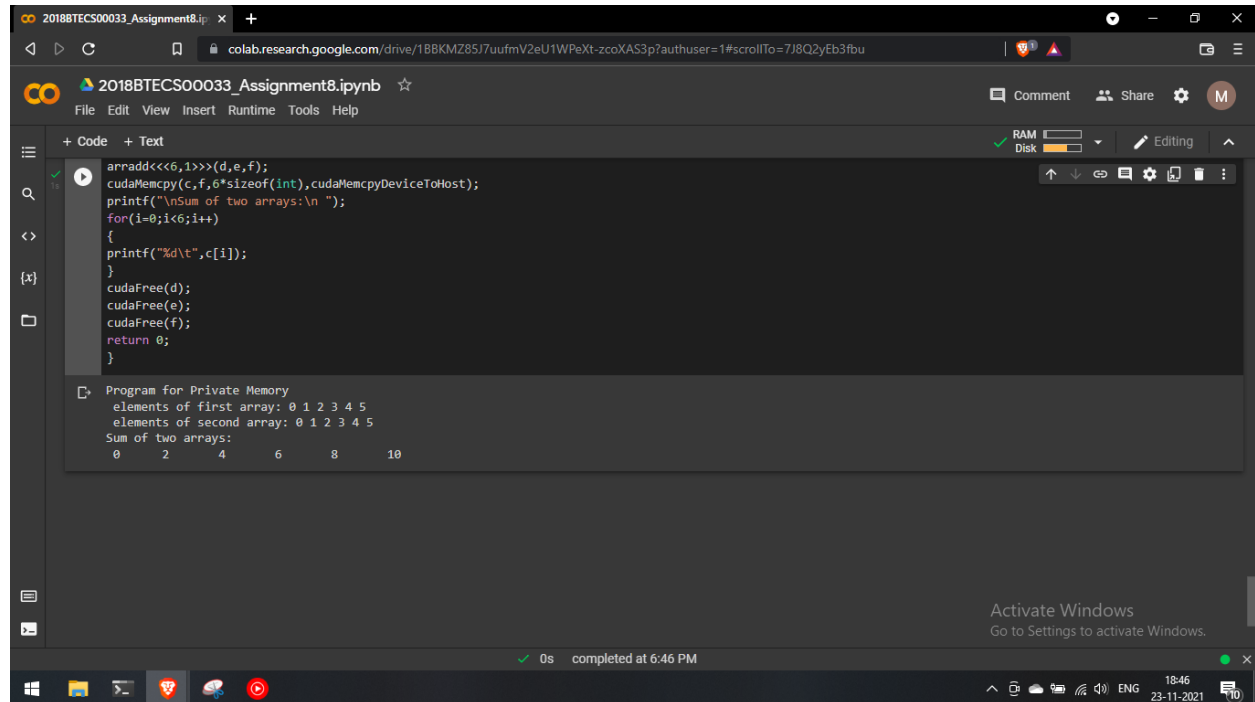
0s completed at 6:23 PM
```

## Information 3: Screenshot of profiling after optimization

### Problem statement 2:

**2. Write a CUDA C program to demonstrate the use of different GPU memories. • Use of private memory. • Use of shared memory. • Use of global memory**

### Screenshot 4:



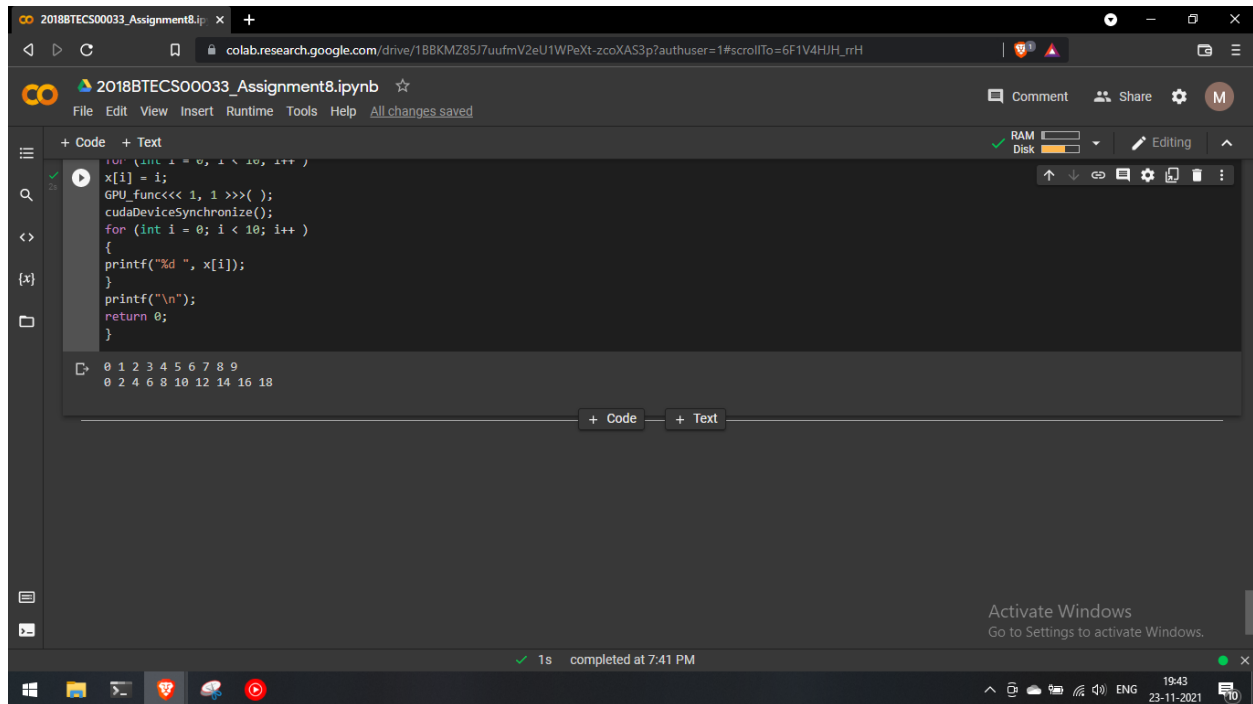
The screenshot shows a Jupyter Notebook interface with a code cell and an output cell. The code cell contains a CUDA C program that uses `arradd` to add two arrays in device memory, then copies the result to the host using `cudaMemcpyDeviceToHost`. The output cell displays the results of the program, showing the elements of the first and second arrays and their sum.

```
arradd<<<6,1>>>(d,e,f);
cudaMemcpy(c,f,6*sizeof(int),cudaMemcpyDeviceToHost);
printf("\nSum of two arrays:\n ");
for(i=0;i<6;i++)
{
    printf("%d\t",c[i]);
}
cudaFree(d);
cudaFree(e);
cudaFree(f);
return 0;
}
```

Program for Private Memory  
elements of first array: 0 1 2 3 4 5  
elements of second array: 0 1 2 3 4 5  
Sum of two arrays:  
0 2 4 6 8 10

### Information 4: Program demonstrating Private memory

### Screenshot 5:

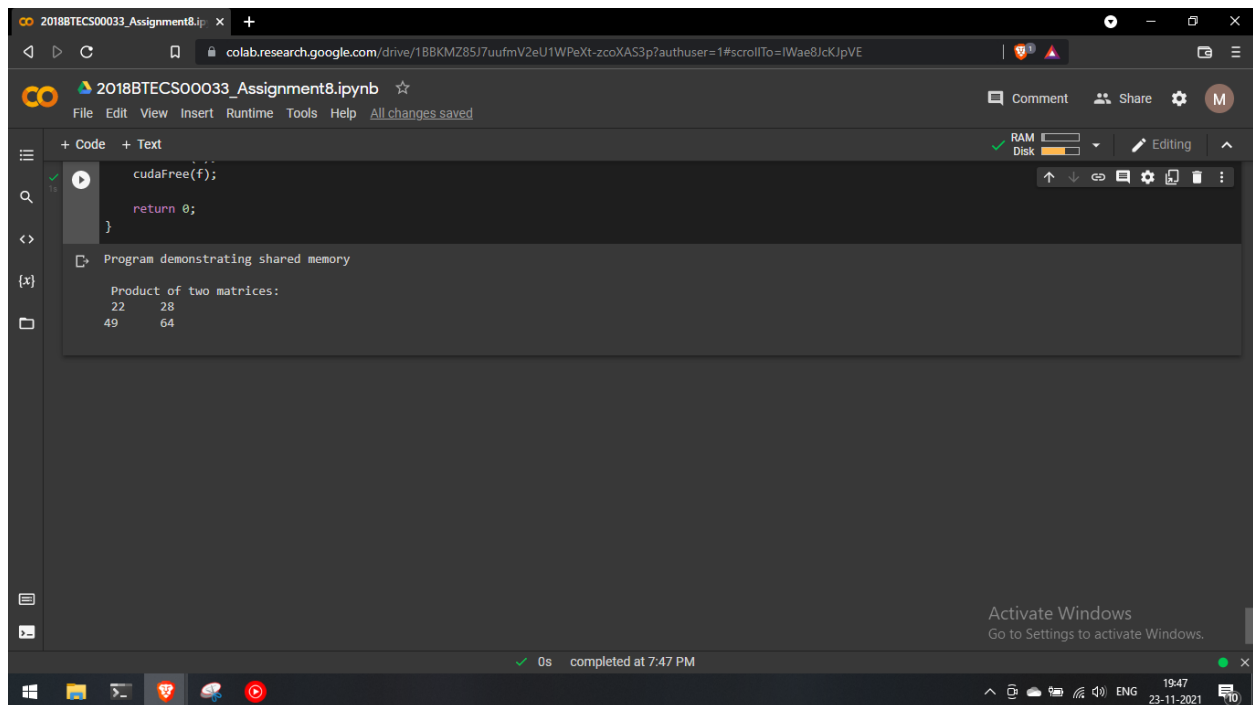


```
for (int i = 0; i < 10; i++)
{
    x[i] = i;
    GPU_func<<< 1, 1 >>>( );
    cudaDeviceSynchronize();
    for (int i = 0; i < 10; i++)
    {
        printf("%d ", x[i]);
    }
    printf("\n");
    return 0;
}
```

0 1 2 3 4 5 6 7 8 9  
0 2 4 6 8 10 12 14 16 18

### Information 5: Program demonstrating Global memory

### Screenshot 6:



```
cudaFree(f);
return 0;
}
```

Program demonstrating shared memory

Product of two matrices:  
22 28  
49 64

### Information 6: Program demonstrating shared memory

Github Link: <https://github.com/g-mahendra/HPC LAB ASSIGNMENTS>