RHODES UNIVERSITY
*Where leaders learn*

Computer Science Honours

# Cuda Programming
# Prac 5

**Shared Memory with checkSmemSquare.cu**
**(follows Lecture 15)**

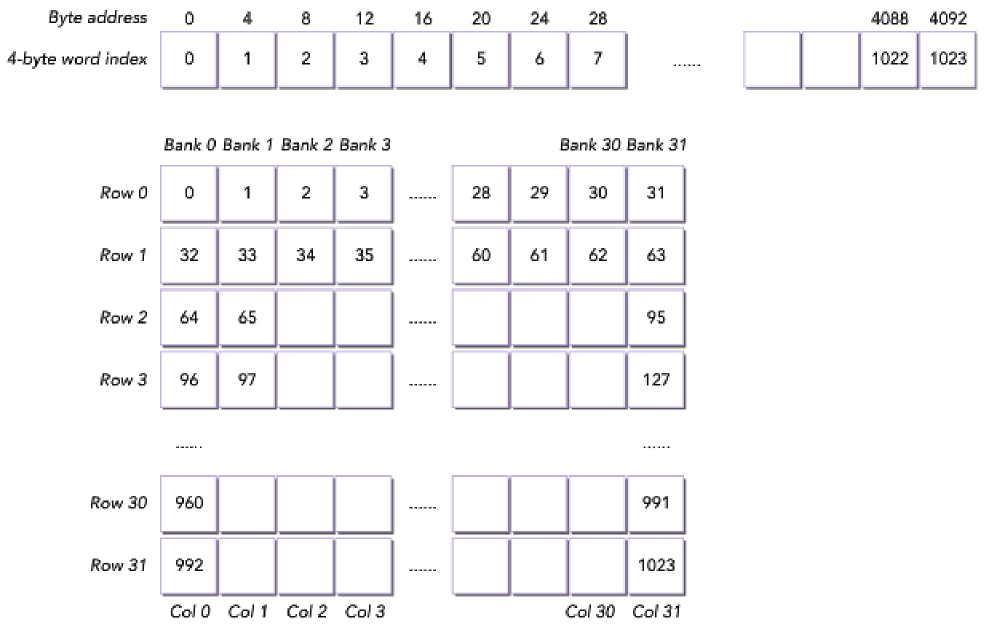
**Instructions** **Marks: 20**
*Due date: 12 noon on 28th February 2020*

Please submit a hardcopy of the results and the information obtained for Tasks 1 to 3.
By submitting your assignment via the RUConnected link, you declare that the assignment submitted is entirely your own work, unless noted otherwise.

## Aim
Investigating the cause of bank conflicts in shared memory.

## Tasks
Download checkSmemSquare.cu on RUConnected.

This program uses a 2D square matrix in shared memory with various memory store and load access patterns. The top figure shows the actual arrangement of 1D data and the bottom figure shows the logical 2D shared memory view with a mapping between 4-byte data elements and banks.



This program attempts to show the difference between row-major vs. column-major access; static vs. dynamic shared memory declarations; and memory padding vs. no memory padding using kernels that are launched with 32 x 32 block size and 1 x 1 grid size.

## Task 1. Accessing row-major vs. column-major

Kernel `setRowReadRow()` has two simple operations:

a. writes the global thread indices to a 2D shared memory array in row-major order
b. reads those values from shared memory in row-major order and stores them to global memory

This means there are three memory operations in the kernel:

- one store operation on shared memory
- one load operation on shared memory
- one store operation on global memory

Note the `__syncthreads()` that is used to ensure that all threads have stored their data to shared memory, before this memory is written to global memory.
Now consider kernel `setColReadCol()`. This kernel writes and reads the data to/from shared memory in column-major order.

Profile each of the kernels to obtain answers to the following questions:

- Which kernel is faster and why?
- Which, if any, suffers from bank conflicts and why?

## Task 2. Write row-major and read column-major

Kernel `setRowReadCol()` executes the same operations as in Task 1, but performs shared-memory writes in row-major order and shared memory reads in column-major order.

Profile the kernel and report on any bank conflicts.

## Task 3. Padding statically declared shared memory

Kernel `setRowReadColPad()` adds an extra column to the 2D shared memory allocation to avoid bank conflicts.

Profile the kernel and report on any bank conflicts.

## Task 4. Comparing the performance of the Square Shared Memory kernels

Report on the runtimes for each of the kernels (obtained via profiling) using 32x32 block size as well as whether there are any bank conflicts with respect to either read or write operations.
To fill in the last column of the assessment, reduce the block size to 4x4, and print the output of the kernels, by calling the program with a runtime argument (any value other than zero).

**Prac 5 -- Shared Memory**

**Name:** **Marks: 20**

| **Kernel executed** | **GPU execu-tion time [ ]** | **Load bank conflicts [Y/N]** | **Store bank conflicts [Y/N]** | **Reason for conflicts** | **Output for 4x4 block size** |
|---|---|---|---|---|---|
| setColReadCol | | | | | |
| setRowReadRow | | | | | |
| setRowReadCol | | | | | |
| setRowReadColPad | | | | | |

## Profile information for Tasks 1-3: