**Introduction to Operating systems Assignment Summary 2:**

The keywords that need some explanation to understand the paper better

**Data races**: when multiple threads of a single process are running concurrently data race condition is said to occur if one of the threads is trying to write a memory shared by the threads and that both threads are operating without acquiring exclusive lock on the shared resource, leaving the memory in the inconsistent state. When these conditions hold the results of the computation will be non-deterministic depending upon the order in which the threads operate on the shared resource.

**Stream multiprocessors**: CUDA allocates one warp to each stream multiprocessors. Stream multiprocessors are different from the general cpus in the sense of having thousands of registers for partitioning the registers among the threads, multiple caches, warp schedulers for fast context switching and also to transfer the next instruction to the warp ready to be executed. They also have special functions for integer and floating point operations.

**Important points to keep in mind before going ahead...**

* GPUs have a complex memory hierarchy arrangement.
* PTX has implementation of named producer and consumer barriers present in NVIDIA GPU. Inline PTX assembly code is made available to the CUDA GPUs as they do not have the support of named barriers.
* We call the kernel well synchronized if all the barriers executed without going into a deadlock state and re initialized to be used further (because they are limited hardware resource!).

1. INTRODUCTION

The main objective of the paper is to provide operational semantics for verifying the correctness of the warp specialized functions that make use of producer consumer named barriers for synchronization that can be run on the gpus. The authors of the paper describe that there are no effective algorithms to verify the kernels written in non-traditional manner. The authors also claim that they have found some bugs in the approaches already proposed.

In the beginning of the paper the authors provide a very high level understanding of the GPU architecture by explaining about the thread blocks, Cooperative thread arrays, warps (a set of 32 threads) which was already covered by me in the last summary.

They explain in traditional model of programming one CTA is assigned on computational task where as in warp specialized kernels each warp (a partition in the CTA) is assigned a different task. Which facilitates a huge amount of data to be present on the chip at the time of execution and a better synchronization approach can be employed. In traditional model thread wide blocking was used but in warp specialized kernels named barriers (a limited resource implemented in hardware) can be used.

They explain why it is tedious task to verify the correctness of the warp specialized functions. It is because of possibility of

* Deadlocks: To make sure that named barriers do not cause a dead lock situation
* Data races: As thousands of threads are running data races can occur. At any time such a situations should be at the best prevented.
* Safe barrier recycling: As the named barriers are limited, they have to be re-initialized after each use.

1. An example of a program that goes into a deadlock.

1 global void launch bounds (64,1) example deadlock(void) {

2 assert(warpSize == 32);

3 assert(blockDim.x == 64);

4 int warp id = threadIdx.x / 32;

5 if (warp id == 0) {

6 // bar.sync (barrier name), (participants);

7 asm volatile(”bar.sync 0, 64;”);

8 asm volatile(”bar.arrive 1, 64;”);

9 } else {

10 asm volatile(”bar.sync 1, 64;”);

11 asm volatile(”bar.arrive 0, 64;”);

12 }

13 }

In the above program we can observe that there are two warps that are running in parallel at presumably different speeds. The warp with warp id 0 blocks itself at named barrier 0 with the sync operation waiting for the warp with warp id 1 to register itself at named barrier 0 by calling arrive but that never happens because the warp with warp id 1 itself blocks at named barrier 1 waiting for the warp with id 0 to signal by arriving at named barrier 1 which never happens!

MOTIVATING KERNELS

Authors present to two important properties of the warp specialized kernels:

1. The reuse of the named barriers in the same warp group execution: To show this the authors provide the execution sequence of the operations in the warp sets which are data parallel and warp wide. The most important condition to reuse the named barriers would be to check happens before condition for the warps participating in the named barriers by taking a look at operations flow char.
2. The assumption that the flow of the programs written is static that is can be determined at compile time: They ensure that the assumptions hold true for all the warp specialized functions and that its very common to have programs whose execution flow is deterministic.