IOS: Summary Part 2.

2.1

Current Available frameworks to program GPUs are based on data parallel programming model. CUDA is used as a proxy as it is the only model which supports named barrier primitives required necessary to construct warp specialised kernels.

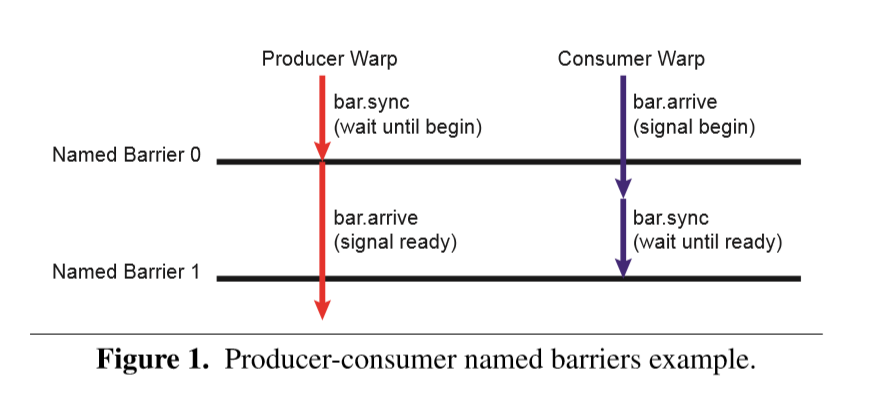
When a CUDA kernel is launched, it creates a collection of CTA or thread blocks. Each block consists of 1024 threads. A hardware inside a GPU dynamically allocates a thread block to one of the Streaming Multiprocessor. **A streaming multiprocessor consists of thousands of registers which can be shared by threads, multiple caches, warp schedulers and execution cores for integer and floating point operations.** For communication between the threads, CUDA provides software managed shared-memory. CUDA provides syncthread primitives to coordinate this shared memory. Synchronization within a CTA is supported by CUDA and no other synchronization as only threads within a CTA run concurrently.

The CTA is further divided into blocks of 32 threads called warps and the hardware within the SM does scheduling at the granularity of warps. Threads within a warp execute the same instruction in case of a branch instruction SM first executes the not taken branch with the threads of the taken-branch masked off. And then executes the taken branch with the former masked off. This serial execution of different branches within a warp is called branch divergence and is primarily a source of performance degradation.

2.2 Warp specialisation and named barriers

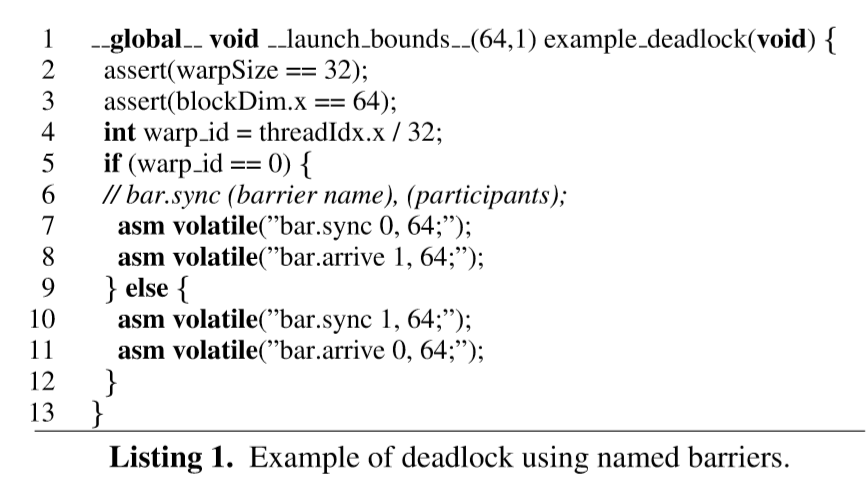
CudaDMA, an extensible API for efficiently managing data transfers between the on-chip and off-chip memories of GPUs. CudaDMA enables the programmer to decouple the size and dimensionality of the data from the size and dimensionality of the computation, improving both programmability and performance. Decoupling is achieved by specializing warps into compute warps and DMA warps. Compute warps are solely responsible for performing computation while DMA warps are solely utilized for moving data between on-chip and off-chip memories. The CudaDMA API provides the abstractions and synchronization primitives necessary for warp specialization.

Similar to CUDA based on data parallel kernels, warp-specialised kernels also communicate using shared memory but this communication is asymmetric. A producer consumer-based synchronization model is followed. The DMA warps act as producers and the compute warps act as consumers. NVIDIA GPU support 16 physical named barriers numbered from 0-15 to achieve this synchronization. Two instruction sync and arrive are used by the threads to block and register the arrival of warps at the barrier respectively. When a producer warp arrives at a barrier it blocks with the help of sync instruction. When a consumer warp arrives, it uses arrive instruction to signal the producer that it is ready to consume and the producer can write. The consumer then resumes with its task. After some point it reaches the second barrier and executes the sync instruction waiting for producer to finish. Then producer arrives and executes the arrive instruction letting execution to continue.



After the named barrier completes it is reinitialized immediately.

Each use of a named barrier is called a generation. Different generation of named barriers can have different participants.



In the above code if two threads with one belonging to warp with id 0 and one from another warp then both of them execute the blocking operation sync, first one on barrier 0 second one on barrier 1 and they get blocked waiting for each other to arrive and signal the unblocking call. Hence, one thread waits for other in a cyclical fashion resulting in a deadlock.

Named barrier introduce the possibility of a deadlock. Which is one of the major drawbacks.

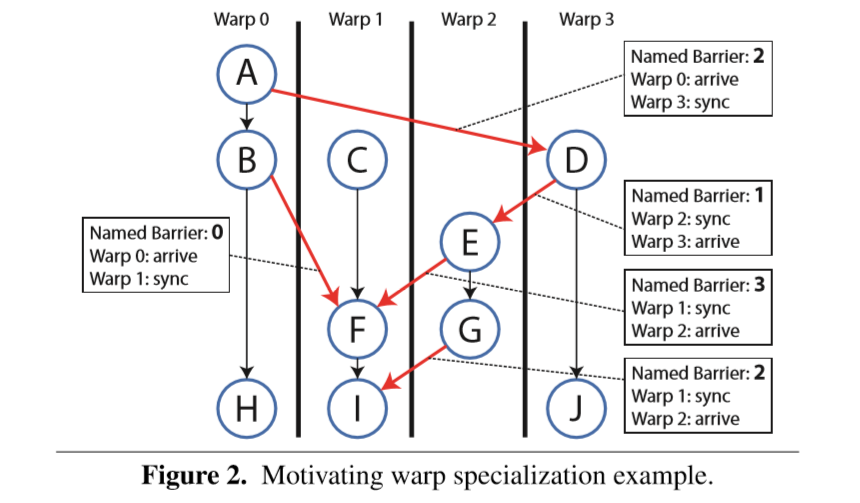
2.3 Motivating kernel

An example to explain the need for verification of warp-specialised kernels:

Heptane chemistry kernel emitted by the Singe DSL compiler.

For instance:

Computation represented as a static dataﬂow graph is mapped on to four warps



When a data flow edge crosses the walls within the warps (Red edges) data is communicated through shared memory.

There are two important properties of this kernel:

1. Barrier reuse: The barrier 2 is reused. To avoid conflicts between two generations of the barrier, a happens before relationship is established between the paths D->E->G and

D>E->F->I, among barriers 1 and 3, to ensure a happened before relationship between the warps of previous generation and the next generation of barrier 2.

1. Static Synchronization: The presence of named barriers constraints the synchronization, so that the synchronization is deterministic and known at compile time. It is static in the sense that thread executes a straight-line of code which does not have any arbitrary control flow statements, and branches. Most of the code written for warp-specialised code is written this way hence the paper gives the operational semantics to by taking the above assumption.